UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E DE ESTATÍSTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERRAMENTAS DE DESENVOLVIMENTO PARA MICROCOMPUTADORES DEDICADOS BASEADOS EM PROCESSADORES 80386/80486

por

Isâac Benjamim Benchimol

Dissertação submetida à Universidade Federal de Santa Catarina
para a obtenção do grau de mestre em Ciência da Computação

Prof. Rogério Cid Bastos, Dr.
Orientador

Florianópolis, maio de 1995

FERRAMENTAS DE DESENVOLVIMENTO PARA MICROCOMPUTADORES DEDICADOS BASEADOS EM PROCESSADORES 80386/80486

ISAAC BENJAMIM BENCHIMOL

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE

MESTRE EM CIÊNCIA DA COMPUTAÇÃO

ESPECIALIDADE SISTEMAS DE COMPUTAÇÃO E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO.

Prof. Rogério Cid Bastos, Dr.
Coordenador do Curso

BANCA EXAMINADORA:

Prof. Rogério Cid Bastos, Dr.
Orientador (Presidente)

Lúcia H. M. Pacheco

Profª. Lúcia Helena Martins Pacheco, M.Sc.
Co-orientadora

Prof. Vitório Bruno Mazzola, Dr.

Prof. Murilo Silva de Camargo, Dr.

"Let's swim to the moon
Let's climb through the tide
Penetrate the evening
That the city sleeps to hide"

(Jim Morrison)

A meus pais

Benjamim Isaac Benchimol (*in memorian*) e
Gracília Figueiredo Benchimol

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família. A minha mãe *Gracília* que me educou e jamais poupou esforços para edificar minha formação. Ao meu irmão *Aaron*, pelo apoio e companheirismo.

Ao professor *Hermann Adolf Harry Lücke*, meu orientador durante quase a totalidade deste trabalho. Embora não esteja mais presente entre nós, os ensinamentos que ele nos deixou como grande pesquisador e personalidade humana, estarão sempre em nossas lembranças.

Ao professor *Rogério Cid Bastos*, que assumiu a minha orientação na fase final do trabalho, pelas suas sugestões e contribuições. À profesora *Lúcia Helena Martins Pacheco*, pela valiosa colaboração dispensada a este trabalho.

Às amigas *Angelita* e *Anita*, pelo apoio e encorajamento durante os momentos difíceis e pela permanente prontidão na revisão deste texto. Ao amigo *Sidney*, companheiro de estudos e intermináveis debates sobre a arquitetura do 80386.

Aos amigos *Sílvio*, *Marcelo* e *Cleto*, companheiros de moradia na minha estada em Florianópolis. Eles foram os responsáveis pelas horas de bom papo e distração.

À *Vera Lúcia* (Verinha), pela atenção e amizade dispensadas durante a minha permanência na UFSC.

Ao meu tio *Saul Benchimol* por acreditar e apostar na minha capacidade.

À UFSC e CNPq que propiciaram as condições necessárias para a realização deste trabalho.

AGRADECIMENTO ESPECIAL

Este trabalho é decorrente das idéias do Prof. Hermann Adolf Harry Lücke, o qual, de forma brilhante, o conduziu e orientou na sua quase totalidade. Em virtude de seu falecimento precoce, o Prof. Hermann não pode acompanhar a apresentação e defesa do mesmo. Entretanto, a ele são creditados todos os acertos e virtudes da presente dissertação.

Prof. Rogério Cid Bastos

Isaac Benjamim Benchimol

RESUMO

Este trabalho propõe o desenvolvimento de microcomputadores dedicados, baseados nos processadores Intel 80386/80486 operando no modo protegido. Esta escolha visa possibilitar a utilização de um PC 386/486 compatível como microcomputador de desenvolvimento dos softwares de sistema e de aplicação, bem como os compiladores mais conhecidos para as linguagens de programação Assembly e C. Entretanto, tais compiladores são voltados para o desenvolvimento de softwares tendo o MS-DOS como ambiente de execução e, consequentemente, não suportam a programação no modo protegido nem tampouco as configurações específicas de ROM, RAM e E/S de um típico ambiente dedicado. Neste trabalho foi desenvolvido um conjunto de ferramentas que objetivam auxiliar o desenvolvimento, o teste e a depuração de softwares para um ambiente dedicado, que utiliza os recursos da arquitetura dos processadores Intel 80386/80486 disponíveis no modo protegido, além de usufruir dos benefícios de um processador real de 32 bits. A funcionalidade das ferramentas é mostrada na simulação de um ambiente dedicado num microcomputador PC 386 compatível, através da construção de dois aplicativos simples.

ABSTRACT

This work proposes the development of embedded microcomputers based on Intel 80386/80486 processors, operating in protected mode. This choice allows a PC 386/486 compatible to be used as the application and system software development computer, as well as the most popular Assembly and C compilers. However, such compilers are designed in order to develop software for the MS-DOS execution environment, and therefore the protected mode programming neither the specific RAM, ROM and I/O configurations of a typical embedded environment are not supported. A set of tools was developed to aid in software development, test and debugging for an embedded environment, that uses the Intel 80386/80486 protected mode architecture features, in addition to make use of a real 32 bits processor benefits. The way the tools can be used is shown through the construction of two simple applications, using a PC 386 compatible computer.

SUMÁRIO

Pág.

Pág.

Pág.

GLOSSÁRIO

GDT	Tabela de descritores global. É um array de descritores de segmento que podem ser acessados por todas as tarefas de uma aplicação. Pode conter até 8192 descritores.
LDT	Tabela de descritores local. É um array de descritores de segmento que podem ser acessados apenas por uma tarefa específica. Pode conter até 8192 descritores.
IDT	Tabela de descritores de interrupção. É um array de descritores de *gates* que associa cada vetor de interrupção ou exceção com um procedimento ou tarefa de serviço para o evento associado. Pode conter até 256 descritores.
TSS	Segmento de estado de tarefa. É um tipo especial de segmento onde o contexto de uma tarefa é guardado quando de sua interrupção, ou de onde o contexto de uma tarefa é restaurado para o seu prosseguimento.
Descritor	Estrutura de dados de 8 bytes que fornece ao processador o tamanho e endereço de um segmento, bem como informações de controle e status.
Seletor	Identifica um descritor de segmento, especificando uma tabela de descritores e um descritor dentro da tabela.
Gate	É um tipo especial de descritor que fornece proteção para o controle de transferência entre segmentos executáveis de diferentes níveis de privilégio.
Aliases	Dois descritores são aliases se eles definem segmentos com o mesmo endereço linear. São usados para fornecer características alternativas de um segmento para tarefas diferentes, ou fornecer uma característica para o sistema operacional e outra para um programa aplicativo.
Queue	Fila FIFO que permite guardar código de instruções previamente buscadas na memória, e de onde as mesmas são lidas para execução.

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

A utilização de microcomputadores tem crescido a cada ano em todas as áreas da atividade humana. Hoje é praticamente impossível conceber uma empresa, por menor que ela seja, sem os recursos que um microcomputador pode oferecer.

Os microcomputadores de uso geral possuem suas arquiteturas padronizadas. A arquitetura PC é uma das mais populares e tem disponível uma grande variedade de software para praticamente todas as áreas de aplicação. Entretanto, há um grupo de microcomputadores, talvez menos conhecido, mas de grande importância econômica, que possui utilização funcional única, e executa aplicações que necessitam de hardware e softwares específicos. Os microcomputadores adaptados para estes tipos de aplicações são chamados de *microcomputadores dedicados.* Neste trabalho será usada a palavra **dedicado**(s) toda vez que for feita referência a microcomputador(es) dedicado(s).

Como exemplo de dedicado cita-se um forno de microondas. Um microcomputador que controla um forno de microondas não necessita de monitor, nem tampouco de dispositivos acionadores de disco rígido ou flexível, mas sim de uma interface apropriada para obter comandos e apresentar informações para o operador, de modo a executar a tarefa solicitada sob um desempenho satisfatório. O forno de microondas é um exemplo simples. Um exemplo mais complexo é a impressora a laser que exige maior rapidez no processamento, devendo apresentar um alto desempenho.

Os dedicados têm na automação uma das suas fontes de aplicação mais complexas. Nesta área, são encontradas aplicações de controle de equipamentos de chão de fábrica e controle de processos. Porém, há características que são comuns independentemente do tipo de aplicação. Do ponto de vista do programador os seguintes componentes estão sempre presentes: uma CPU, memórias ROM e RAM, e espaço de endereçamento para as entradas e saídas (E/S) [BRO94].

1.1. Motivação e Objetivos do Trabalho

Os microprocessadores Intel 80386 e Intel 80486 constituem poderosas CPUs para microcomputadores de uso geral, sendo utilizados na arquitetura PC-AT. Os recursos existentes no modo protegido de operação destes microprocessadores, satisfazem as exigências de sistemas de software de alto desempenho. Recursos como segmentação e paginação, proteção de memória, capacidade de multitarefa, recursos para depuração e compatibilidade com programas desenvolvidos para o 8086, além de memória cache e co-processador integrados na pastilha, no caso do 80486, justificam suas utilizações como CPUs de sistemas complexos. Além disto eles são bastante conhecidos, de fácil disponibilidade e seus preços estão em constante declínio.

Outra motivação para a escolha dos microprocessadores 80386/80486 para este trabalho é a crescente utilização de microprocessadores de 32 bits em várias aplicações dedicadas.

A produção de microprocessadores de 32 bits em 1993 chegou a 140 milhões de peças e estima-se que este número dobrará até 1997. Um estudo do mercado [ELE94] revela que até 1997, 73% da produção mundial de microprocessadores de 32 bits vai ser utilizada em sistemas dedicados, ou seja, quase 3/4 de mais ou menos 280 milhões de peças vão equipar toda a sorte de equipamentos que não sejam microcomputadores de uso geral. Este estudo também mostra (figura 1.1) a utilização dos microprocessadores de 32 bits em 4 áreas de aplicação — periféricos, comunicação, eletrônica de consumo e indústria/outros — em 1993 e a projeção para 1997. Nota-se que há uma tendência acentuada de crescimento na utilização destes microprocessadores em áreas como eletrônica de consumo. Isto demonstra a importância econômica que os dedicados estão adquirindo.

Neste trabalho será utilizado o termo "80386" referindo-se tanto ao processador Intel 80386 quanto ao processador Intel 80486. Porém o termo "80486" será utilizado apenas para o processador Intel 80486.

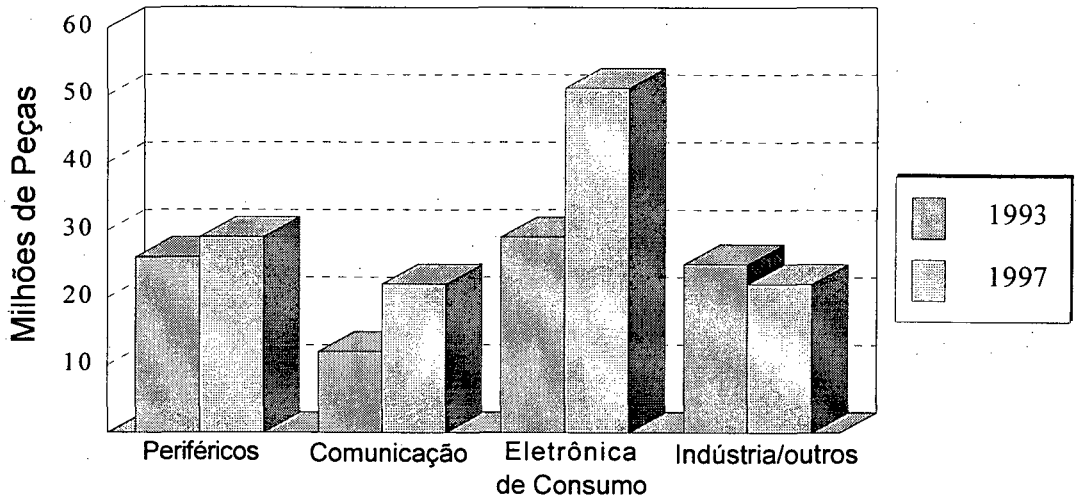


Figura 1.1 - Gráfico demonstrativo de utilização de microprocessadores de 32 bits em aplicações dedicadas[ELE94].

O objetivo principal deste trabalho é desenvolver ferramentas que auxiliem no desenvolvimento, teste e depuração de softwares para dedicados, que possuem como CPU, o 80386 operando no modo protegido. Além disso, pretende-se permitir a utilização de um microcomputador PC 386 compatível e todo o ferramental disponível para o sistema operacional MS-DOS — editores, compiladores (para as linguagens C e Assembly), ligadores — para desenvolver softwares para os dedicados.

Embora a CPU do microcomputador de desenvolvimento seja a mesma do dedicado, o desenvolvimento de software apresenta algumas particularidades, devido ao fato dos compiladores comuns serem voltados para o ambiente MS-DOS, enquanto o ambiente dedicado possui características de memória e E/S específicas. O fato do software ser desenvolvido no modo protegido particulariza ainda mais o seu desenvolvimento, pois estes mesmos compiladores não suportam este modo de operação. O presente trabalho apresenta este problema e propõe alternativas de solução.

Há disponível alguns kits de desenvolvimento de software para o modo protegido. A Intel possui o seu kit — o **System Builder** — que fornece compatibilidade com

programas escritos em linguagem C e Assembly. Entretanto, o System Builder é uma ferramenta cara, custa alguns milhares de dólares e é complicado de usar. Outra empresa que fornece um pacote para desenvolvimento de aplicações de 32 bits no modo protegido é a SSI (Systems & Software, Inc.). Este pacote inclui ferramentas para a ligação e realocação de arquivos objetos gerados por compiladores para linguagens C e Assembly; um gerador de arquivo imagem para EPROM; uma biblioteca para substituir as funções dependentes do DOS e ferramentas para a depuração remota no microcomputador dedicado. Todo o pacote custa em torno de US$ 3.300,00 incluído o suporte técnico. Ou seja, é uma ferramenta cara, sem considerar ainda a dificuldade de utilização.

Este trabalho objetiva transpor estas dificuldades desenvolvendo ferramentas de fácil utilização e compreensão. Para alcançar este objetivo, faz-se necessário reunir e dominar o conhecimento dos recursos do 80386 aplicáveis à construção de ferramentas que auxiliem no desenvolvimento de softwares para dedicados. A capacitação em programação no modo protegido deve ser alcançada. Pretende-se que as ferramentas sejam utilizadas em softwares escritos no modo protegido com as linguagens C e Assembly.

Uma dificuldade foi detectada de imediato durante o início dos estudos: toda a documentação disponível se resumia a alguns manuais da Intel [INT87, INT90a, INT90b, INT91] — meramente descritivos e contendo informações espalhadas em diversos capítulos — , alguns livros [SCA88, CRA87, SIN90, FER90, SEG92, MIL90] sobre a arquitetura e programação do 80386 e alguns artigos [WIL90a, WIL90b, WIL90c, ALB90, MAR89, GRE89, FRI90a, FRI90b, KNO92, LEI93, BAR92, BUR94] que davam exemplos de programação sendo especialmente úteis na elucidação de algumas artimanhas comumente utilizadas no complicado modo protegido desse processador. Os códigos que acompanhavam esses artigos foram boas fontes de consulta embora não esgotassem o assunto. Uma boa parte das informações, esclarecimentos e sucessos foram alcançados após muita persistência na utilização dos recursos.

1.2. Apresentação do Trabalho

O trabalho está dividido em 7 capítulos, iniciando com esta introdução. O capítulo 2 faz uma abordagem sobre dedicados e seus componentes, o hardware e o software. As características e dificuldades encontradas no desenvolvimento de software para um ambiente dedicado são abordadas. Ainda neste capítulo, é discutida a solução geralmente adotada pelos kits de desenvolvimento existentes comercialmente.

No capítulo 3 é apresentada uma proposta de hardware e de software de sistema para dedicados. As ferramentas são propostas como elementos do software de sistema. Este capítulo justifica a escolha do 80386 e do hardware a ser utilizado como CPU do sistema. Os processadores 80386 e 80486 e a placa-mãe da arquitetura AT 386/486 são descritos brevemente. Ao final do capítulo é feita uma análise comparativa entre as ferramentas propostas e os kits existentes comercialmente.

O capítulo 4 é voltado para o projeto das ferramentas. Aqui são apresentadas as vantagens, características, dificuldades e soluções encontradas para o desenvolvimento de software para dedicados utilizando um microcomputador de uso geral 386 com sistema operacional MS-DOS. O procedimento adotado para execução e teste do software dedicado no próprio microcompuatador de desenvolvimento é mostrado.

O capítulo 5 descreve a implementação das ferramentas. O capítulo 6 descreve a utilização das ferramentas na simulação de um ambiente dedicado utilizando um microcomputador PC 386 compatível. O capítulo 6 também descreve as etapas do procedimento adotado para o desenvolvimento de dois aplicativos exemplos, nos quais as ferramentas foram utilizadas. As diferenças entre a abordagem que utiliza o PC de desenvolvimento simulando o dedicado e a que executa o software diretamente a partir da EPROM na placa é discutida. As conclusões são apresentadas no capítulo 7 e a bibliografia no capítulo 8.

O apêndice A reúne as macros que foram construídas e testadas durante a realização deste trabalho. As macros constituem uma parte das ferramentas implementadas para o desenvolvimento de software para dedicados.

O apêndice B reúne as características do processador 80386 quando operado no modo protegido. As informações foram selecionadas e compactadas de várias fontes de consulta e são resultado de muito estudo e experimentações práticas realizadas em softwares escritos no modo protegido. Os recursos integrantes do modo protegido de operação e a maneira de utilizá-los são mostrados de forma simples e sucinta. O apêndice B é uma das fontes de consulta para o desenvolvimento deste trabalho.

2. Microcomputadores Dedicados

2.1. Introdução

Microcomputadores dedicados são parte da vida cotidiana. De fato, desde utensílios domésticos até acionadores de discos e impressoras a laser, os dedicados estão sempre presentes em algum equipamento manuseado diariamente. Mesmo os não-usuários de computadores de uso geral utilizam involuntariamente dúzias deles sem perceberem o fato [COO91].

Os microcomputadores dedicados recebem este nome justamente por não constituirem um microcomputador de uso geral, mas sim parte de equipamentos, como telefones sem fio, automóveis, etc, que são dedicados a controlar tarefas específicas. Estes equipamentos são chamados *sistemas dedicados* [TZO93].

Tais sistemas possuem características específicas que envolvem tanto o hardware quanto o software dedicado. Uma grande área de aplicação para estes sistemas é a automação. Dentro desta área pode-se encontrar desde sistemas mais simples como uma máquina de lavar até sistemas que necessitam de um desempenho maior como uma impressora a laser. A máquina de lavar possui, por exemplo, um comando de temporização da função lavar e monitora a temperatura da água. A impressora a laser deve receber os dados e controlar o mecanismo de impressão a laser. O microcomputador dedicado é a parte do sistema que controla estas funções dedicadas.

Comparando um microcomputador dedicado em relação a um de uso geral, constata-se as seguintes diferenças [KOO89]:

- Tamanho e peso - O microcomputador dedicado apresenta tamanho e peso menores, porque ele é apenas parte de um equipamento. Isto é possível devido principalmente a sua menor complexidade.

- Potência - O microcomputador dedicado requer um menor consumo de potência para não comprometer o restante do sistema. Pelo fato de possuir menor complexidade, automaticamente os dedicados consomem menos potência.

- Ambiente de operação - Geralmente o microcomputador dedicado opera em condições extremas e deve lidar normalmente com choques, vibrações, calor e frios intensos, e até radiação.

- Desempenho computacional - Dependendo do sistema, o microcomputador dedicado necessita de um grande desempenho, principalmente se for de tempo-real.

- Confiabilidade e robustez - O microcomputador dedicado deve ter uma maior confiabilidade. Um erro de operação ou de especificação pode causar consequências catastróficas.

2.2. Hardware Dedicado

O hardware de um sistema dedicado é restrito àquele estritamente necessário à função dedicada [TZO93]. Processador e dispositivos especiais são utilizados. Diferentemente de computadores de uso geral, os dedicados geralmente não acompanham periféricos como monitores, unidades de disco rígido ou flexível e impressora. Em vez destes, o dedicado deve possuir uma interface apropriada, de modo a obter comandos e apresentar informações para o operador.

A CPU é a parte mais importante do hardware dos dedicados. Como CPU pode-se utilizar um *microcontrolador* ou um *microprocessador*. Um microcontrolador é um dispositivo que agrega em uma única pastilha o processador, a memória e as funções auxiliares como temporizadores, controlador de interrupção, interface serial e outros. É preciso atenção para escolher entre utilizar um ou outro. Um dos principais critérios para

fazer a opção é o desempenho exigido. Embora os microcontroladores sejam poderosos, eles não conseguem acompanhar o poder dos microprocessadores quando, por exemplo, for necessário utilizar memória adicional e funções auxiliares externas. Nestes casos, os microcontroladores gastam ciclos a mais que os microprocessadores equivalentes [COO91]. Muitas vezes, os microcomputadores dedicados precisam de todo o poder que lhes é designado, mesmo que isso implique no custo e na complexidade de utilização de um microprocessador.

A escolha do microprocessador também deve ser estudada com cuidado. Os microprocessadores de 32 bits estão ganhando terreno e já estão sendo utilizados com frequência em aplicações dedicadas, e a tendência é utilizá-los cada vez mais nestes tipos de aplicações. Além disso, aplicações na área de automação industrial já exigem todo o poder dos microprocessadores de 32 bits.

Dentre os microprocessadores, há aqueles projetados para utilização em microcomputadores de uso geral e os chamados *microprocessadores dedicados* que apresentam arquitetura voltada para utilização em sistemas dedicados. Os processadores Intel 80376 e Intel 386EX são exemplos de microprocessadores dedicados de 32 bits. O 386EX é um modelo mais recente. Ele possui o microprocessador 386SX como núcleo e tem integrado uma série de componentes em sua pastilha. A figura 2.1 mostra os blocos com os componentes internos do386EX [INT94].

Outros microprocessadores são largamente utilizados como CPUs de microcomputadores de uso geral. Na arquitetura PC, a Intel lidera o mercado equipando-a com a família de processadores x86. O 80386 e o 80486 são microprocessadores de 32 bits poderosos, são fáceis de serem encontrados e apresentam preço baixo. Isto faz a diferença com relação aos microprocessadores dedicados, que não são bem conhecidos e portanto não são tão fáceis de serem encontrados. Consequentemente seus preços são mais elevados.

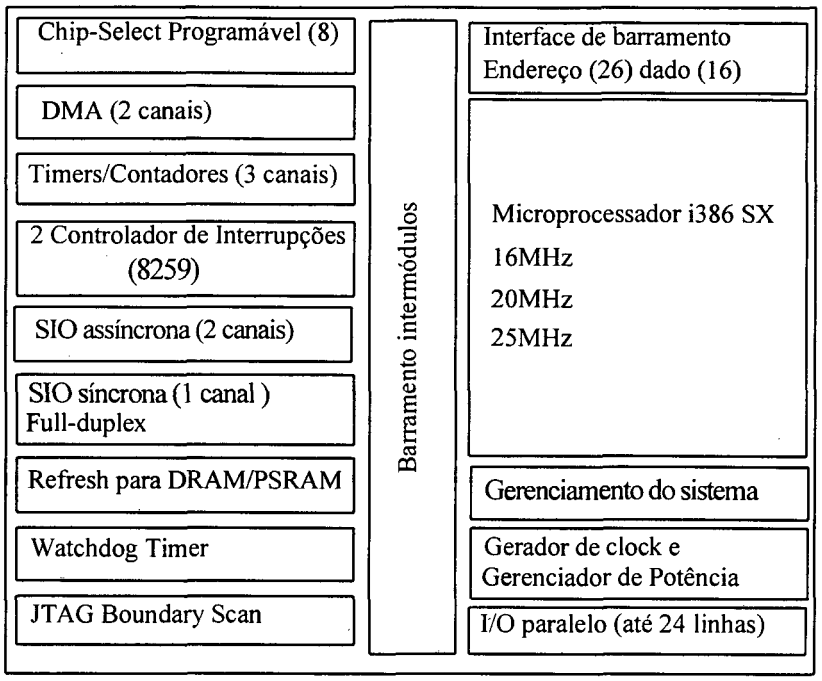


Figura 2.1 - Componentes internos do microprocessador dedicado 386EX [INT94].

A figura 2.2 mostra um exemplo de microcomputador dedicado reunindo numa placa um microprocessador de uso geral, memórias RAM e EPROM, barramento de E/S e interface de comunicação serial e paralela. Através do barramento de E/S é possível conectar componentes especiais necessários à função dedicada, como por exemplo, microcontroladores, portas para E/S de sinais digitais e placas de conversores analógico-digital (A/D) e digital-analógico (D/A) para realizar operações de E/S de sinais analógicos.

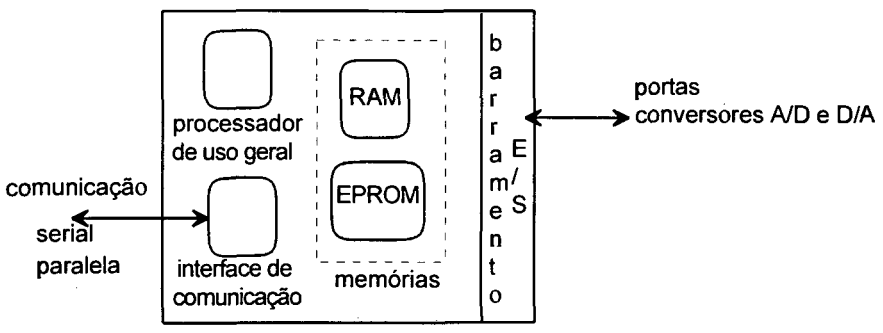


Figura 2.2 - Exemplo de microcomputador dedicado.

2.3. Software Dedicado

O software que é executado no sistema dedicado é chamado de software dedicado e é composto pelos softwares de aplicação e software de sistema.

2.3.1. Software de Aplicação

Num sistema dedicado o software de aplicação é responsável pela execução da função dedicada. Os elementos que compõem o software de aplicação são *entrada de dados, operação e saída de dados* (figura 2.3).

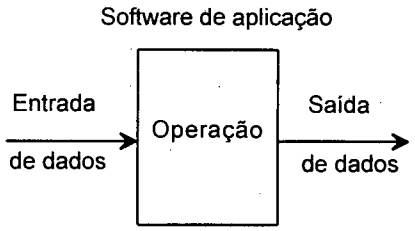


Figura 2.3 - Elementos do software de aplicação.

A *entrada de dados* recebe os dados que serão processados pelo elemento *operação.* Os dados são sinais que representam grandezas elétricas externas ao sistema. Tais grandezas podem ser do tipo analógicas ou digitais. Quando uma grandeza elétrica analógica for lida é necessário convertê-la para uma cadeia de sinais digitais — com valores de 0V ou 5V, no caso da lógica TTL — através do conversor A/D. No caso das grandezas elétricas digitais é necessário apenas garantir que elas estejam dentro dos valores aceitáveis da lógica utilizada. Cada grandeza elétrica lida representa uma grandeza física envolvida com o sistema. Exemplos de grandezas físicas são: pressão, velocidade, temperatura, vazão, posição de um motor, posição de um braço mecânico, etc.

A *operação* do software de aplicação é constituída por todo o processamento interno necessário para executar a função dedicada. Em grandezas de origem analógicas são aplicadas funções algébricas e em grandezas de origem digital, funções booleanas.

Uma vez terminada a operação sobre os dados de entrada, o software de aplicação deverá promover a *saída de dados* de volta ao mundo externo. Nesta etapa, os sinais de saída que necessitarem ser convertidos em sinais analógicos, por exemplo, para o controle de atuadores, serão levados ao conversor D/A que se encarregará desta operação. Os sinais de saída que serão tratados digitalmente deverão garantir uma potência suficiente para o propósito utilizado.

2.3.2. Software de Sistema

O software de sistema engloba várias funções num sistema dedicado. Primeiramente ele deve inicializar todo o hardware presente, configurando-o de acordo com as necessidades. Aqui incluem-se por exemplo os controladores de interrupção, portas paralelas e portas seriais. Funções básicas de E/S também fazem parte do software de sistema. Elas facilitam a operação de entrada e saída de dados externos. Num microcomputador de uso geral essas funções são fornecidas pela BIOS e pelo Sistema Operacional [DUN90, SCH90].

Uma característica do software de sistema é quanto à capacidade de realizar comunicação com outros computadores. Dependendo do seu uso, essa característica pode ser essencial. Sistemas que não suportam comunicação são conhecidos como *standalone.* No caso em que a comunicação é parte integrante do sistema, ela pode ser realizada via interface serial RS-232 ou via interface de rede ETHERNET. A capacidade de comunicação permite a comunicação com outros computadores, a interação com o operador e ainda pode ser explorada na fase de desenvolvimento.

Na fase de desenvolvimento a interface de comunicação permite acoplar-se ao dedicado um microcomputador PC de uso geral emulando um terminal. Já que os

dedicados geralmente não possuem monitor nem teclado, o terminal pode substituí-los auxiliando na depuração (*debug*) e realização de testes durante o desenvolvimento do sistema. Para efetuar a comunicação deve-se utilizar funções de E/S que utilizem a interface serial para enviar dados a serem amostrados no monitor e para receber comandos digitados pelo teclado do terminal. Desta forma o sistema pode interagir com o programador ou projetista.

As possibilidades de interação podem ser ampliadas por um *carregador*. Este elemento tem a função de armazenar o software de aplicação — todo ou parte —, extensões, dados e parâmetros recebidos via serial na memória RAM do dedicado. O carregador é responsável pela determinação dos endereços onde o programa/dados serão armazenados e executados. Com o uso de um carregador pode-se enviar, por exemplo, um arquivo, rotinas, ou parâmetros para o sistema.

Recursos para depuração são desejáveis e extremamente importantes. Pode-se usar um PC para controlar a execução no dedicado remotamente ou simplesmente para a monitoração de seu status.

Ao final do desenvolvimento, dependendo da aplicação, deve-se ter, se não todos, pelo menos alguns elementos do software de sistema gravados em EPROM. A tabela 2.1 mostra cinco situações possíveis para o software de aplicação, o carregador, o software de comunicação e o depurador quanto ao destino da execução (memória EPROM ou RAM). A inicialização não entrará nesta discussão pelo fato dela sempre ser executada em EPROM.

	Software de aplicação	Carregador	Software de comunicação	Depurador
1	EPROM	-	-	-
2	EPROM --> RAM	-	-	-
3	EPROM	EPROM	EPROM	EPROM
4	RAM	EPROM	EPROM	RAM
5	Simulador de EPROM			

Tabela 2.1 - Situação de execução no sistema dedicado.

A tabela 2.1 explica-se a seguir, de acordo com a numeração apresentada na sua primeira coluna:

1. O software de aplicação é armazenado em EPROM, enquanto os outros elementos inexistem, caracterizando um sistema do tipo *standalone* com impossibilidade de passagem de parâmetros ou extensões do sistema, além de não suportar ferramentas de depuração. É usado em sistemas que não precisam de alterações nem parametrizações.

2. É similar à primeira situação com a diferença que o software de aplicação é gravado em EPROM, mas transferido para memória RAM a fim de ser executado no momento da inicialização. Essa alternativa facilita a programação e melhora o desempenho do sistema já que a memória RAM é mais rápida que a EPROM.

3. Possui tanto o software de aplicação quanto os demais elementos embutidos em EPROM, permitindo a comunicação com outros sistemas, inclusive a conexão de um terminal para depuração. Essa alternativa também permite transferir o software de aplicação para a RAM durante a inicialização do sistema.

4. O carregador e o software de comunicação estão gravados em EPROM e são responsáveis pela recepção do software de aplicação e das rotinas de depuração provenientes do terminal e pelos respectivos armazenamentos em memória RAM. Esta alternativa representa o modelo de teste do sistema, que vai ser executado no hardware dedicado em condições reais de operação, evitando que se "queime" uma EPROM a cada alteração do software de aplicação.

5. Essa alternativa utiliza um simulador de EPROM. Neste caso, os softwares de sistema e de aplicação são carregados independetemente do microcomputador dedicado e é executado na memória RAM do simulador como se fosse

executado na própria EPROM do dedicado. O simulador também é direcionado para o desenvolvimento e teste do sistema. Ele será explorado com mais detalhes na seção 4.2.2.1.

2.4. Desenvolvimento de Software para Dedicados

O processo de desenvolvimento de software dedicado apresenta pelo menos duas características que o diferem daquele utilizado para desenvolver softwares convencionais [COO91]. A primeira diferença decorre do fato dos dedicados serem desprovidos de dispositivos periféricos e softwares requisitados para o desenvolvimento de programas, tornando necessária a utilização de um microcomputador de uso geral para realizar esta tarefa. A segunda diferença é que o software dedicado, salvo alguns casos, é desenvolvido em paralelo com o hardware dedicado [TZO93].

O fato do desenvolvimento ser realizado num microcomputador de uso geral, implica na utilização de ferramentas voltadas para determinados sistemas operacionais. Isto ocasiona uma série de problemas que deverão ser considerados durante o desenvolvimento. O principal deles é decorrente da utilização de compiladores *e* ligadores para uma linguagem de programação qualquer. Considerando o desenvolvimento em um ambiente MS-DOS, e portanto, utilizando compiladores para este sistema operacional, os seguintes problemas devem ser considerados:

- Compiladores DOS geram código dependente do MS-DOS, que é o sistema operacional para o qual foi construído. Como o ambiente dedicado não possui sistema operacional, pelo menos com a funcionalidade daqueles utilizados por microcomputadores de uso geral, a dependência do código não pode existir;

- Os arquivos gerados pelos compiladores são realocáveis, ou seja, eles podem ser executados em qualquer endereço. O MS-DOS possui um carregador de programas que os aloca conforme a memória RAM disponível. Em um

ambiente dedicado, os endereços são fixos, ou absolutos, e uma vez determinados pelo projetista, não mudam a cada execução;

- O ambiente MS-DOS permite o carregamento e a execução de programas apenas em memória RAM, porém o dedicado utiliza as memórias ROM e RAM para executar sua tarefa. Os compiladores tradicionais não permitem separar códigos e dados para as configurações específicas de um ambiente dedicado;

- Por fim, deve ser ainda considerada a indisponibilidade de um microcomputador de desenvolvimento que utilize a mesma CPU do dedicado. Neste caso, a utilização de um *cross-assembler* é obrigatória.

Os problemas apresentados dificultam o desenvolvimento de software principalmente em relação à fase de testes. As operações de E/S são um exemplo. O microcomputador de uso geral não possui as E/S específicas do dedicado, ou ainda, se as possuir, elas podem não estar disponíveis. Outro agravante é com relação aos endereços de execução dos softwares de sistema e de aplicação. Estes fatores fazem com que diminua a produtividade dos programadores durante o teste do sistema, o qual torna-se especialmente crítico com esta situação, obrigando-os a utilizar ferramentas e técnicas alternativas muitas vezes subótimas [TZO93].

Uma técnica que pode aumentar a produtividade no desenvolvimento é a simulação do hardware do sistema no ambiente de desenvolvimento, que embora demande algum tempo para torná-lo suficientemente realístico, é útil para se observar o comportamento geral do sistema. Para tal, existe uma ferramenta chamada *In-Circuit Emulator* (ICE), que constitui-se essencialmente de um microcomputador dedicado, contendo hardware e software capazes de emular totalmente as características funcionais de uma dada CPU. O ICE é conectado à unidade a ser testada através de um cabo especial, que encaixa fisicamente no soquete da CPU a ser emulada. Para realizar a conexão, a CPU é retirada de seu soquete e substituída pelo cabo do ICE. A partir daí, todas as atividades da CPU são realizadas pelo ICE de forma transparente ao software e

hardware. Com a utilização de um depurador é possível interromper a execução do software sob determinadas condições e perceber o comportamento do hardware. Além disso, é possível mapear partes ou toda a memória da unidade a ser testada para o ICE, possibilitando que o código a ser testado seja carregado diretamente para ele. Isso permite "enganar" a unidade sob teste fazendo-a acreditar que a sua estrutura de memória resida no ICE. É possível, também, mapear o espaço de endereçamento de E/S para o ICE, possibilitando a depuração do software antes mesmo da disponibilidade do hardware. Tudo isso ainda tem a vantagem de não precisar gravar um byte sequer em EPROM economizando tempo e esforços.

2.4.1. Ferramentas Comerciais para Desenvolvimento de Softwares Dedicados

Há ferramentas no mercado voltadas para a construção de aplicativos dedicados no modo protegido do 80386 sob a forma de kits de desenvolvimento. Geralmente estes kits são direcionados para as linguagens C e Assembly, as mais indicadas para este tipo de aplicação. Porém, muitos deles restringem sua utilização a compiladores específicos de 32 bits como METAWARE C/C++ e WATCOM C/C++ [SSI93, SSI94, CON94].

A figura 2.4 mostra os componentes (software e hardware) geralmente encontrados em um kit de desenvolvimento para dedicados. A seguir são descritos os componentes conforme a numeração apresentada nessa figura.

1. **Compiladores** - São oferecidos opcionalmente, mas há obrigatoriedade de utilizá-los a fim de manter a compatibilidade com o restante do kit. Os compiladores para a linguagem C são geralmente versões de 32 bits menos conhecidas e, consequentemente, requerem adaptação a um novo ambiente de programação.

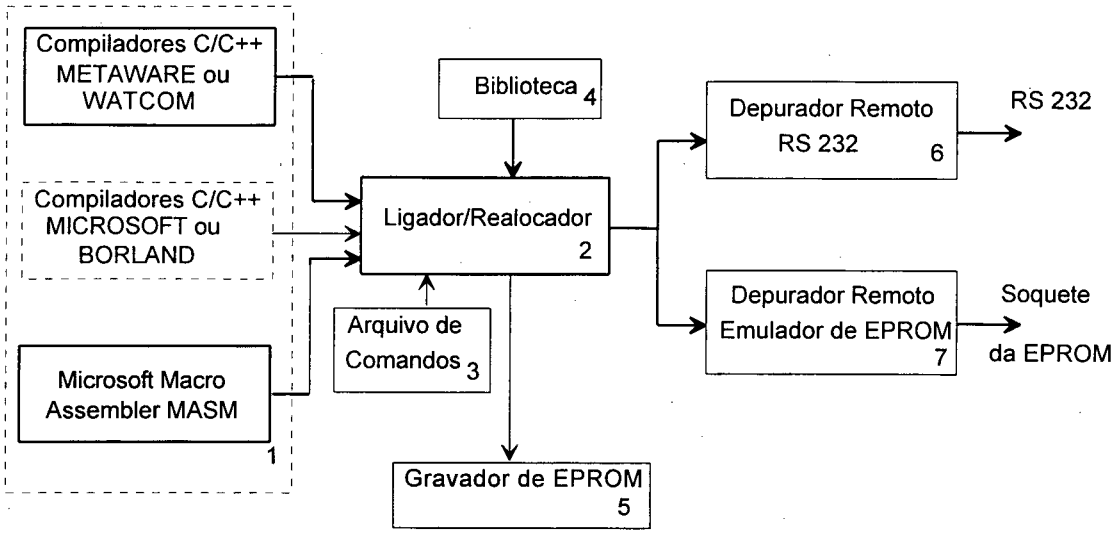


Figura 2.4 - Componentes de um kit de desenvolvimento para dedicados [SSI93, SSI94].

2. **Ligador/Realocador** - Aceita como entrada arquivos objetos gerados pelos compiladores compatíveis, e gera em sua saída arquivos absolutos com formatos suportados pelos depuradores remotos e pelo gravador de EPROM. Este bloco realiza a realocação de endereços e a construção do sistema através das informações contidas num arquivo de comandos.

3. **Arquivo de Comandos** - Contém todas as informações relacionadas à realocação e à construção do sistema. Uma linguagem de especificação pré-definida é utilizada para a construção das tabelas do sistema (GDT, LDT e IDT) e dos segmentos de estado de tarefas (TSS's). É possível também definir endereços para cada segmento do programa.

4. **Biblioteca** - A ausência de um sistema operacional no hardware destino, requer a substituição de funções dependentes do DOS presentes no *startup* e nas funções de biblioteca com código voltado para o dedicado. Esta ferramenta suporta funções independentes do DOS dispensando sua gravação em EPROM.

5. **Gravador de EPROM** - Esta ferramenta também é oferecida opcionalmente. Ela recebe o arquivo absoluto do Ligador/Realocador e o grava numa EPROM

especificada. Porém é preciso verificar que formatos de arquivos são aceitos pelo Gravador de EPROM.

6. **Depurador Remoto para RS 232** - Aceita formatos de arquivos específicos contendo informações de depuração, e utiliza a interface serial RS 232 para comunicar com o dedicado. Geralmente é constituido de duas partes: uma interface com o usuário compatível com DOS e um monitor incorporado ao software dedicado, que ocupa até 32Kbytes de EPROM e necessita até 32K bytes de RAM. Utilizando-se esta ferramenta é possível monitorar a execução do software a partir de um PC através da interface com o usuário.

7. **Depurador Remoto para Emulador de EPROM** - Aceita formatos de arquivos específicos contendo informações de depuração, e utiliza um emulador de EPROM conectado ao soquete da EPROM do dedicado. Geralmente acompanha uma interface com o usuário compatível com o DOS e um monitor, mas elimina a necessidade de reservar memória extra em EPROM e RAM para o mesmo.

O preço médio dos kits de desenvolvimento pesquisados vão desde US$ 3.300,00 para pacotes que incluem o depurador remoto para RS 232, até US$ 5.800,00 para pacotes que incluem o depurador remoto para emulador de EPROM. Neste preço está incluído o suporte técnico mas não abrange a licença para múltiplas cópias do monitor que sozinho alcança a cifra de US$ 5.600,00 [SSI94].

3. Proposta e Especificação de Microcomputadores Dedicados Baseados em Processadores 80386/80486

3.1. Introdução

Neste capítulo é apresentada inicialmente uma proposta de hardware para dedicados. Os processadores 80386, 80486 e a placa mãe PC 386/486 da arquitetura PC-AT são abordados brevemente. Em seguida é apresentada uma proposta de software de sistema, o qual engloba, neste trabalho, os recursos que constituem as ferramentas para o desenvolvimento de software dedicado. Ao final do capítulo é feita uma análise entre as ferramentas propostas e os kits de desenvolvimento existentes comercialmente.

3.2. Hardware

O processador escolhido para o dedicado foi o 80386/80486 da Intel. Além disto, optou-se por utilizar não apenas o processador como base do hardware dedicado, mas toda a placa-mãe de um microcomputador PC compatível baseada nestes processadores. As razões para esta escolha são:

- A disponibilidade imediata deste hardware de alto desempenho e confiabilidade; o conhecimento de sua tecnologia e o custo constantemente em declínio. Uma placa 386 sem memória custa em 1995 aproximadamente US$ 100,00;

- A fácil adaptação a processos. Os vários componentes de uma placa-mãe 386/486 são úteis para o sistema dedicado: controladores de interrupção e DMA (Acesso Direto à Memória), temporizadores, interface serial e barramento de E/S na forma de slots. Pode-se utilizar a interface serial para comunicação com um terminal. Para o barramento de E/S há disponibilidade de placas de interface para rede e de conversores A/D e D/A. Estas placas podem

ser utilizadas, respectivamente, para comunicação com uma rede de computadores e para a entrada e saída de sinais analógicos e digitais;

- A possibilidade de poder desenvolver software para o sistema utilizando um microcomputador PC 386 ou 486 compatível, ou seja, a mesma CPU do dedicado;

- A disponibilidade de todo o manancial de software e ferramentas de desenvolvimento existentes para o PC sob o ambiente MS-DOS, incluindo editores, compiladores, ligadores, etc. Porém, os problemas decorrentes da utilização destas ferramentas para o desenvolvimento de software dedicado, devem ser solucionados, tarefa deste trabalho.

Uma desvantagem encontrada com a utilização de uma placa-mãe pronta 386/486 é o tamanho que ela poderia ultrapassar em relação ao limitado a um microcomputador dedicado. Mas as placas-mãe de computadores estão cada vez menores e mais leves e não devem ser problemas para a grande parte dos sistemas dedicados. Além disto, existem placas com toda a funcionalidade de uma placa-mãe, porém, com tamanho reduzido, especialmente fabricadas para utilização na automação industrial. Entretanto, um dedicado construído nestes termos é economicamente viável somente para aplicações que exijam grande desempenho. Aplicações mais simples, que não justificam a construção de placas específicas, podem utilizar microcontroladores. Ainda assim, utilizando-se microcontroladores baseados no 386 todo o software desenvolvido no PC pode ser aproveitado.

3.2.1. O Microprocessador 80386

O microprocessador 80386 lê e processa dados em lotes de 32 bits, possuindo velocidade de processamento muitas vezes superior ao dos seus antecessores, chegando a utilizar um clock de até 40 MHz (versão da AMD). Esse fato já demonstra o seu alto desempenho.

O 80386 pode trabalhar em um dos dois modos de operação: real ou protegido [INT87, INT90a, INT90b, INT91, SCA88, CRA87, SIN90, FER90, SEG92, MIL90]. O modo real é o modo de compatibilidade com os processadores 8086/8088. Neste modo, o 80386 executa as instruções tal como seriam executadas por estes microprocessadores. A principal característica do modo real é com relação à segmentação limitada da memória - cada segmento pode ter tamanho máximo de 64K bytes, sendo que os registradores da CPU armazenam os endereços dos segmentos divididos por 16. Além disto, no modo real é possível endereçar apenas 1M bytes de memória.

No modo protegido esta limitação desaparece. Os segmentos podem ter tamanhos até o limite da memória endereçável, podendo alcançar até 4G bytes. Nesse modo o endereçamento é feito de maneira diferente e descrito na seção 3.2.1.1. Além disso, o 80386 possui internamente uma série de recursos avançados quando operado no modo protegido. Esses recursos, como segmentação, paginação, proteção de memória, capacidade de multitarefa, modo virtual 86 e recursos para depuração, satisfazem a implementação de um sistema de tempo real de alto desempenho.

A segmentação e paginação da memória são formas nas quais o 386 tem de subdividir o espaço de endereçamento linear em segmentos ou páginas. Enquanto uma página de memória tem tamanho sempre igual a 4K bytes, um segmento pode ter tamanho até 4G bytes, sendo limitado somente pela quantidade de memória disponível. A paginação é uma maneira conveniente para o sistema alocar e relocar a memória, já que todas as porções possuem o mesmo tamanho, constituindo-se na chave para a implementação de memória virtual.

O 80386 permite a definição dos endereços de programas em tabelas de descritores de segmento e tabelas de páginas. Estas tabelas permitem que haja a separação de áreas de memória entre tarefas ou código do sistema e tarefas ou códigos dos usuários e garantem a não interferência de códigos e dados das tarefas dos usuários entre si. A proteção de código e dados é essencial para sistemas que necessitem de segurança.

Uma outra vantagem do uso de paginação e segmentação é que elas facilitam o trabalho de realocação de endereços. O deslocamento de um dado dentro de um segmento (*offset*) não muda independente do endereço inicial do segmento. Já a paginação é bastante útil na simulação de endereços "fictícios", pois permite que um programa utilize um endereço lógico para localizar um operando dentro de outro espaço de endereçamento físico completamente diferente - o programa usa um endereço (lógico), mas na verdade está endereçando um outro (físico) na memória.

O recurso de multitarefa possibilita a implementação de sistemas que permitam a execução, suspensão e preparação para execução de diferentes tarefas. O 80386 possui internamente estruturas voltadas para a implementação desse recurso. No chaveamento de uma tarefa para outra, o hardware automaticamente salva o contexto da tarefa atual e restaura o contexto da próxima tarefa que vai executar [INT87, INT90a, INT90b, INT91].

O modo virtual 86 (VM86) é um submodo do modo protegido, onde programas escritos para o microprocessador 8086 podem ser executados normalmente como se fossem tarefas de um sistema implementado no 80386, fazendo uso de quase todos os recursos da operação em modo protegido. Usando o modo virtual 86 é possível executar, por exemplo, as funções do DOS dentro do modo protegido [BAR92].

O recurso de depuração compreende registradores especiais que definem endereços para a ocorrência de *breakpoints*, onde o processador gera uma exceção quando o programa acessar essas posições de memória. Assim, é possível executar uma rotina de depuração sob determinadas condições e sob determinados endereços. Execução passo-a-passo também é possível.

3.2.2. O Microprocessador 80486

O microprocessador 80486 possui todas as características mencionadas acima pertencentes ao microprocessador 80386 somando-se, ainda, uma porção de memória cache de 8K bytes e co-processador aritmético integrados na mesma pastilha.

A memória cache constitui uma área de memória estática de baixo tempo de acesso em relação à memória RAM principal de um sistema. O subsistema de cache interno do 80486 permite a execução de código mais rapidamente, já que o microprocessador efetuar ciclos de buscas de instruções no cache em vez de na memória principal do sistema. As instruções mais prováveis de serem executadas, aquelas da sequência natural de execução, são carregadas na cache para serem lidas posteriormente. É possível ainda estabelecer-se que porções do código são carregadas na cache e trocar o seu conteúdo somente quando for desejado [INT87, INT90a, INT90b, INT91]. Porém, deve-se considerar a situação onde um trecho de programa que não esteja na cache seja solicitado para execução. Neste caso, o tempo para executá-lo será maior, causando problemas se ele for determinístico. A memória cache constitui um fator extremamente importante em sistemas com requisitos críticos de tempo.

Outro recurso presente no 80486 é o co-processador aritmético. Ele é responsável pela execução de operações aritméticas com ponto flutuante, muito comuns em programas que lidam com números decimais, planilhas, aplicações de CAD (projeto auxiliado por computador), aplicações industriais tais como controle de processos, cálculo de trajetórias etc. Tais aplicações são executadas no 80486 com um desempenho bastante superior do que em outros microprocessadores desprovidos desse recurso, ou com a presença dele numa pastilha externa.

Além desses recursos, o 80486 é capaz de operar com multiprocessamento - possibilidade de operação de mais de um microprocessador no sistema. É útil para sistemas complexos e, especialmente, críticos em tempo.

Assim pode-se constatar que os recursos avançados disponíveis no 80486 adequam-se perfeitamante a sistemas com características de multitarefa e proteção, sistemas críticos em tempo e sistemas que apresentam grandes quantidades de cálculos em ponto flutuante. Em aplicações na automação industrial, onde geralmente é exigido um alto desempenho, como por exemplo, em controle de equipamentos de chão de

fábrica e em controle de processos, estes processadores são uma ótima opção de utilização.

Esse conjunto de recursos disponíveis no modo protegido do 386/486 serão descritos mais detalhadamente no apêndice B deste trabalho.

3.2.3. A Placa 386/486 da Arquitetura PC-AT

A placa-mãe AT-386/486 compatível possui uma série de componentes que facilitam a sua adaptação a um microcomputador dedicado. Os slots de expansão, controladores de DMA (2), controladores de interrupção (2), temporizadores (3), além de soquete disponível para co-processador, no caso do 80386, estão presentes nestas placas.

Nos slots de expansão pode-se utilizar uma série de placas disponíveis no mercado para a arquitetura AT: placas conversoras D/A e A/D, placas de rede, portas serial e paralela, etc. Os controladores de DMA podem ser utilizados para a transferência de grandes quantidades de dados para/da memória de/para um dispositivo de E/S. Os controladores de interrupção e os *timers* permitem o atendimento de eventos assíncronos do processo e podem ser programados conforme as necessidades do sistema. As portas serial e paralela podem ser utilizadas para comunicação com um terminal para troca de dados durante a operação ou mesmo durante o desenvolvimento do sistema para realizar testes. No caso do 80386, que não tem embutido um co-processador aritmético, é possível conectá-lo ao soquete disponível para aumentar o desempenho de programas que trabalham com muitos cálculos, especialmente em operações com ponto flutuante.

É notória a diminuição do tamanho dessas placas ao longo dos anos. Isso facilita ainda mais o seu acoplamento a sistemas dedicados com restrição de espaço físico.

3.3. Software Dedicado

3.3.1. Software de Sistema

Nesta seção será feita uma proposta de elementos para o software de sistema de dedicados. As ferramentas de desenvolvimento, objeto deste trabalho, serão incorporadas ao software de sistema, e serão propostas no decorrer da descrição dos elementos.

3.3.1.1. Inicialização

O software de sistema se encarregará primeiramente de inicializar e configurar o hardware da placa-mãe. A inicialização do sistema deverá ser feita no modo real de operação que é o modo em que o 80386 entra quando é ligado. Alguns componentes podem ter suas inicializações aproveitadas de uma BIOS de um PC 386 compatível mas outros deverão ter suas configurações alteradas para serem utilizados com o sistema dedicado. A reprogramação do PIC (controlador de interrupções programável), por exemplo, é importante, pois algumas interrupções padronizadas pelo PC coincidem com exceções geradas no modo protegido. Exceções são geradas toda vez que ocorrer alguma situação anormal em tempo de execução, por exemplo, violação de proteção geral, acesso a páginas não presentes na memória, divisão por zero, etc [INT87, INT90a, INT90b, INT91]. No modo real as exceções são restritas à divisão por zero e à exceção de passo único (passo-a-passo).

A principal diferença entre os modos real e protegido do 80386 é com relação à forma de endereçamento. No modo real os registradores de segmento (16 bits) armazenam os valores dos endereços de segmento divididos por 16. Assim qualquer acesso à memória é feito deslocando esses valores de 4 bits para a esquerda e somando-se o *offset* do operando. No modo protegido, os endereços dos segmentos são armazenados em descritores em uma das tabelas do sistema: a GDT (tabela de descritores globais), a LDT (tabela de descritores locais) e a IDT (tabela de descritores de

interrupção). Estas tabelas contêm descritores para cada segmento utilizado num programa. Neste modo, os registradores de segmento contêm na sua parte visível apenas um número de descritor em uma das tabelas. Este índice chama-se *seletor*. Portanto, para endereçar um segmento no modo protegido, basta carregar um registrador de segmento com um seletor e o 80386 automaticamente vai buscar no descritor correspondente o seu endereço base [INT87, INT90a, INT90b, INT91]. Isto faz com que os programas não manipulem diretamente com o endereço de base dos segmentos.

A necessidade das tabelas faz com que elas precisem ser montadas antes da passagem para o modo protegido. Durante a inicialização, as tabelas do sistema devem ser montadas bem como os descritores dos segmentos de código e dados que serão acessados no modo protegido. Os descritores referentes ao software de aplicação também devem ser construídos. No caso em que as tabelas já estejam armazenadas em EPROM, o software de sistema precisa copiá-las para a RAM, pois ela pode ser atualizada posteriormente.

O uso das tabelas de descritores permite definir os endereços de cada segmento do programa já durante o desenvolvimento. Uma vez feito isso deve-se apenas garantir que os segmentos sejam carregados na memória exatamente no mesmo endereço definido pelos seus descritores. Se, de outra forma, os endereços forem definidos apenas durante a execução é preciso atualizar os descritores correspondentes com os endereços utilizados logo após o carregamento.

As primeiras ferramentas propostas serão utilizadas durante a inicialização dos sistema. O **construtor** tem como objetivo automatizar a construção da GDT, LDT, IDT e dos TSS's. O **realocador interno** terá a função de manter coerentes os endereços de carregamento dos segmentos com o endereço base no respectivo descritor.

Para a realocação referente ao modo real, é proposto um **realocador externo** que solicitará os endereços de realocação para segmentos em memória RAM e EPROM.

O capítulo 4 mostra os meios de realizar a realocação de segmentos no modo real e no modo protegido. As tabelas e outras estruturas utilizadas no modo protegido estão apresentadas com detalhes no apêndice B.

3.3.1.2. Comunicação

O sistema possuirá capacidade de comunicação. Ele poderá conectar-se com um terminal para troca de dados. Isto é importante na execução de programas desenvolvidos com compiladores para o MS-DOS. Para tal, algumas funções do MS-DOS relacionadas à E/S serão utilizadas. As funções de entrada pelo teclado e de saída pelo vídeo serão adaptadas para utilizar a interface serial RS 232C. Outras funções como as de gerenciamento da memória de massa em forma de discos rígidos ou flexíveis também podem ser adaptadas para carregar um arquivo pela interface serial, uma vez que o sistema dedicado não possuirá estes dispositivos. Não serão implementadas outras funções como por exemplo um sistema de troca de mensagens.

3.3.1.3. Funções de E/S

Estas funções podem ser implementadas tais como no MS-DOS: utilizando o modo real da família x86 ou fazendo uso dos 32 bits do modo protegido. Na primeira alternativa, as funções devem ser executadas no modo virtual 86 (VM86) onde há compatibilidade com o modo real, mas, se a decisão for utilizar a segunda alternativa, deve-se escrevê-las conforme dita o modo protegido do 80386, não precisando deixá-lo para executá-las [ALB90]. Como o objetivo do trabalho é ter ao final, o sistema pronto em EPROM funcionando no microcomputador dedicado, as funções de E/S que serão utilizadas obedecerão a segunda alternativa — adaptação ao modo protegido (já que o microcomputador dedicado não terá disponível o MS-DOS nem suas funções).

Para as funções de E/S será implementado um **Mini-DOS** protegido com as funções mais comuns da INT 21h do MS-DOS. O Mini-DOS terá as E/S redirecionadas

do teclado e vídeo para a interface serial podendo ser utilizado para a comunicação com um terminal conectado pela RS-232. As interrupções implementadas para substituir as similares da BIOS são a interrupção da interface de comunicação serial e a interrupção do relógio.

3.3.1.4. Depuração

A possibilidade de comunicação com um terminal constitui num importante meio de testar e depurar o software durante o desenvolvimento e serve, também, como ferramenta para interação com o operador caso necessário. Comandos de depuração podem ser passados ao dedicado que por sua vez pode responder, enviando o seu status sob determinada condição. É possível, ainda, controlar remotamente, através do terminal, a execução do software de aplicação passo-a-passo.

A ferramenta para depuração — o **depurador** — será implementada utilizando os recursos da arquitetura do 80386.

3.3.1.5. Carregador

Com a opção de comunicação, pode-se permitir que softwares de aplicação, rotinas, comandos e parâmetrros sejam enviados ao dedicado a partir do terminal. Neste trabalho será implementado um **carregador**, que terá a função de receber e armazenar na memória os segmentos dos softwares de aplicação recebidos pela interface serial. O **carregador** também será responsável pela alocação de descritores, em uma tabela de sistema, para cada segmento recebido garantindo a coerência com os endereços de carregamento.

3.4. Análise Comparativa Entre as Ferramentas Propostas e os Kits Comerciais

As diferenças básicas entre as ferramentas propostas e as disponíveis nos kits comerciais são apresentadas a seguir.

- **Compilador** - As ferramentas propostas permitem a utilização dos compiladores mais conhecidos do mercado como Microsoft C, Borland C e Microsoft Macro Assembler (MASM). Os kits comerciais são geralmente voltados para compiladores de 32 bits, menos conhecidos e mais caros;

- **Ligador** - Pode-se utilizar na solução apresentada, os ligadores que acompanham os compiladores citados acima. Os kits fornecem um ligador especial que engloba as funções de ligação e realocação;

- **Realocador** - Para os segmentos no modo protegido, a realocação será realizada internamente pelo software de sistema e efetuado com o auxílio de macros. Para a realocação no modo real será utilizado um realocador externo. Nos kits, a etapa de realocação é realizada juntamente com a ligação dos arquivos objetos. Os parâmetros para realocação como endereços e ordem de segmentos são especificados através de um arquivo de comandos o qual deve ser submetido ao ligador/realocador;

- **Construção do sistema** - A montagem de tabelas e estruturas do sistema é realizada pelo próprio software de sistema com o auxílio de macros. Nos kits, esta etapa também é feita juntamente com a ligação, através de comandos específicos inseridos no arquivo submetido ao ligador/realocador;

- **Recursos para depuração** - Na proposta apresentada, é possível utilizar um terminal conectado ao dedicado pela interface serial a fim de utilizar recursos de depuração, como por exemplo, inserir e retirar *breakpoints* da memória, executar um aplicativo passo-a-passo ou monitorar o estado dos registradores

da CPU. Há recursos de depuração nos kits comerciais que utilizam a interface serial ou um emulador de EPROM para comunicação com o dedicado. Estes recursos incluem uma interface com usuário para PC e um monitor incorporado ao software na EPROM, que ocupa aproximadamente 32K bytes e precisa de mais 32K bytes em RAM;

- **Recurso para carregamento remoto** - Este recurso não foi encontrado nas ferramentas pesquisadas. Nesta implementação será possível enviar ao dedicado um ou mais softwares de aplicação a partir do terminal. O carregador é a ferramenta embutida no software de sistema responsável pela recepção e armazenamento destes, bem como pela alocação de descritores para cada segmento recebido;

- **Funções de E/S** - Este trabalho inclui o desenvolvimento de um Mini-DOS no modo protegido com as funções de E/S mais utilizadas, a fim de fornecer uma maneira de comunicação com o terminal. Os kits pesquisados não suportam a conexão com um terminal, portanto este recurso também não está disponível;

- **Independência do MS-DOS e BIOS** - Os kits comerciais geralmente acompanham uma biblioteca para linguagem C com funções independentes do DOS, que substituem aquelas fornecidas pelo próprio compilador. Isto permite que programas utilizem normalmente as funções da linguagem que são adaptadas para o ambiente do dedicado. Na proposta apresentada é possível apenas utilizar as funções da linguagem C independentes do MS-DOS ou da BIOS e aquelas dependentes que foram implementadas pelo Mini-DOS.

A tabela 3.1 sumariza as diferenças entre as ferramentas propostas e as existentes nos kits comerciais.

Recursos	Kits comerciais [SSI94, CON94]	Ferramentas Propostas
Compiladores	Microsoft MASM MetaWare High C/C++ WATCOM C/386	Microsoft MASM Microsoft C e Borland C
Ligador	Utilizam um especial.	Utilizam o mesmo que acompanha os compiladores.
Realocador	Embutido no Ligador: Permite definir endereços de realocação por um arquivo de comandos.	Embutido no Software de Sistema.Utiliza macros.
Construção do sistema	Embutido no Ligador: Permite a construção das tabelas por um arquivo de comandos.	Embutido no Software de Sistema. Utiliza macros.
Depuração	Opcional. Permite a depuração remota via RS 232 ou com o uso de um emulador de EPROM.	Embutido no Software de Sistema. Permite a depuração remota via RS 232 usando um terminal.
Carregamento Remoto	Não disponível.	Embutido no Software de Sistema. Permite carregar aplicativos via RS 232.
Funções de E/S	Não disponível.	Utiliza um Mini-DOS protegido com E/S para RS 232.
Independência do MS-DOS e BIOS	Acompanham uma biblioteca independente do MS-DOS e BIOS.	Permite utilizar somente funções implementadas pelo Mini-DOS e interrupções da serial e relógio.

Tabela 3.1 - Diferenças entre as ferramentas propostas e os kits de desenvolvimento.

4. Projeto das Ferramentas para Desenvolvimento de Software Dedicado

4.1. Introdução

O desenvolvimento do software para o sistema será feito num microcomputador PC 386 ou 486 compatível, ou seja, a mesma CPU que vai executá-lo ao final. As ferramentas mais conhecidas disponíveis para o MS-DOS serão utilizadas, como os compiladores Microsoft Macro Assembler (MASM), Microsoft C e Borland C, além de ligadores.

A forma como estas ferramentas serão adaptadas para desenvolver software para o microcomputador dedicado no modo protegido será abordada nesta seção. Também o procedimento para a execução e teste do software dedicado no próprio microcomputador de desenvolvimento será abordado. O teste é feito automaticamente já que o hardware de destino é o mesmo. As operações de E/S podem ser simuladas pelo teclado e monitor, entretanto, placas de E/S podem ser instaladas, no barramento do PC de desenvolvimento já que elas são idênticas àquelas utilizadas no microcomputador dedicado, além de utilizar os mesmos endereços de E/S. É claro que isto só será possível se o próprio microcomputador PC não estiver utilizando uma E/S idêntica.

As ferramentas de desenvolvimento serão implementadas em linguagem Assembly, pelo fato desta linguagem ser a mais eficiente para manipular os recursos internos do processador.

Uma primeira decisão a ser tomada diz respeito ao local de execução do software analisando as alternativas da tabela 2.1. Utilizando a alternativa 3 com a opção de transferir o software de aplicação da memória EPROM para RAM permite que a interface serial seja utilizada para receber do terminal apenas eventuais extensões, parâmetros e comandos para depuração, ou ainda carregar aplicações com o objetivo principal de depurá-las. O dedicado trabalha sozinho e não precisa da interface serial para

executar a sua tarefa dedicada. Esse fato justifica a escolha da alternativa 3 já que tem-se menor fluxo de dados pela serial. Além disto, quando o software de aplicação já estiver testado e depurado, não há razão para carregá-lo a cada vez que o dedicado for ligado. Quanto às variáveis utilizadas pelo software, é necessário que estejam em memória RAM a fim de poderem ser manipuladas. Assim, elas devem estar num segmento de dados que também vai ser copiado da EPROM para a RAM.

Para o teste do sistema, a alternativa 5 — o uso do Simulador de EPROM — é a mais prática, pois permite a utilização de exatamente os mesmos endereços de EPROM que o software irá executar ao final.

Uma vez feita a escolha dos locais de execução do software é preciso analisar como resolver um dos problemas envolvidos neste trabalho: *a realocação de endereços*. Realocação é o processo de mudança do endereço físico de um item e o ajuste de todas as referências a esse item. Será visto como o MS-DOS executa esta tarefa para um programa escrito no modo real e a seguir descrever-se-á a proposta para a realocação de programas no modo protegido.

4.2. Realocação no Modo Real (MS-DOS)

Compiladores DOS geram como saída arquivos ".OBJ". Estes arquivos possuem os códigos binários das instruções do programa. O arquivo ".OBJ" é levado a um ligador (LINK) a fim de gerar um arquivo executável ".EXE". Estes arquivos são realocáveis possuindo endereços de segmento definidos com valores temporários pelo compilador durante a montagem. Estes valores são inválidos porque os endereços de execução de qualquer programa só são definidos no momento de seu carregamento pelo MS-DOS e dependem da memória livre do microcomputador [DUN90].

Portanto, um programa ".EXE" gerado pelos compiladores MASM e C possuem endereços inválidos e precisa ser realocado para ser levado à memória e ser executado por um microcomputador.

O arquivo ".EXE" tem um cabeçalho ou módulo de carga com um formato característico que possui as informações necessárias para o carregador do sistema realizar a realocação. No cabeçalho há uma tabela de realocação que contém todos os endereços do arquivo que precisam de realocação [DUN90]. A partir da definição de um endereço inicial de execução, o MS-DOS resolve todos os endereços realocáveis e carrega o programa na memória para ser executado. Esse procedimento é efetuado a cada execução do programa, pois o MS-DOS pode atribuir um endereço de carga diferente dependendo da memória disponivel da máquina. Portanto, o carregador do MS-DOS reúne a função de carregador e realocador.

Para solucionar o problema da realocação de um programa escrito no modo real para ser executado num microcomputador dedicado é preciso, primeiramente, identificar em que endereços o programa será executado. Esses endereços devem ser definidos durante o desenvolvimento. A partir daí é preciso utilizar um realocador para os arquivos ".EXE" gerados pelo LINK. Esse realocador resolve todos os segmentos realocáveis a partir de um segmento inicial solicitado. Desta forma, o programa só necessitará ser carregado na memória do microcomputador dedicado exatamente no endereço inicial definido anteriormente ou gravado em EPROM, se o realocador utilizou o endereço inicial da EPROM. Denomina-se este realocador de *realocador externo*, pois ele será usado independente do software dedicado.

O procedimento acima será utilizado para realocar a porção do software de sistema deste trabalho escrito no modo real - a inicialização. Como essa parte do software residirá na EPROM do microcomputador dedicado, utilizar-se-á no realocador o endereço inicial deste dispositivo. A figura 4.1 mostra as etapas do desenvolvimento de software dedicado escrito no modo real.

O realocador externo será utilizado apenas para levar o software para o microcomputador dedicado já que no microcomputador de desenvolvimento a realocação dos segmentos do modo real será feita pelo próprio MS-DOS.

4.3. Realocação no Modo Protegido

O fato do software dedicado ser desenvolvido em parte no modo real (inicialização) e parte no modo protegido (software de aplicação) torna diferente o processo de definição dos endereços dos segmentos.

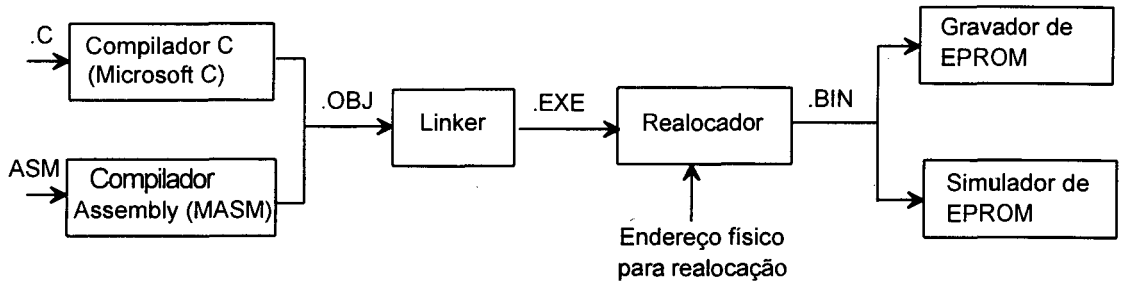


Figura 4.1 - Etapas do desenvolvimento e teste de software escrito no modo real para microcomputadores dedicados.

Com o uso do MS-DOS, o processo de carregamento e realocação dos segmentos é o mesmo, ou seja, o MS-DOS não diferencia entre segmentos no modo real e protegido. Portanto, a utilização do carregador do MS-DOS para definir os endereços dos segmentos não é aplicável a este caso, já que ele não altera os descritores na GDT e LDT para ajustar os endereços de base afim de compatibilizá-los com aqueles efetivamente utilizados para armazená-los e executá-los na memória do computador.

No modo protegido os endereços onde os segmentos serão carregados e executados podem estar definidos antes mesmo da compilação do programa, ou seja, durante o próprio desenvolvimento. As tabelas GDT e LDT contêm descritores que são estruturas que definem todas as informações relacionadas a cada segmento do programa. Os 32 bits de endereço de base do segmento ocupa 4 dos 8 bytes de cada descritor.

Ainda no modo protegido, os registradores de segmento contêm na sua parte visível apenas um número que representa um seletor de um segmento em uma tabela de descritores do sistema, portanto, eles não lidam diretamente com o endereço de um

segmento de memória. Isto faz com que um programa escrito no modo protegido não gere nenhuma referência a ítens realocáveis facilitando a construção de programas para o microcomputador dedicado, que passa a ter referências fixas a segmentos. Por exemplo, a instrução para carregar um registrador de segmento com um seletor, não gera código realocável pelo compilador porque o seletor é um número fixo e não um endereço de segmento como acontece no modo real. Neste modo, a instrução certamente ocasionaria um item realocável pelo compilador.

O uso dos seletores, entretanto, não impede que os segmentos sejam realocados para um endereço qualquer. Para efetuar uma operação de realocação no modo protegido é preciso carregar o segmento para o endereço desejado e ajustar o seu descritor. Porém, esse procedimento só pode ser realizado com a definição de um segmento *alias* para a tabela correspondente, uma vez que o segmento das tabelas não é acessível diretamente pelo programa.

O problema então se resume a garantir que os segmentos sejam carregados na memória exatamente no endereço definido pelo descritor do segmento correspondente. Esses endereços podem ser mudados dinamicamente bastando ter permissão para acessar as tabelas de descritores.

Assim, uma vez determinado o endereço onde um segmento vai ser armazenado e executado, o seu descritor associado em uma das tabelas do sistema deve ter o seu campo de endereço base de 32 bits ajustado. Outra alternativa seria o inverso. Os descritores já poderiam indicar em que endereço carregar cada segmento, cabendo ao carregador realizar a tarefa de ler estes endereços e carregar os segmentos na memória. É importante notar que este procedimento serve não só para os segmentos de código mas também para os segmentos de dados, especialmente aqueles que contém variáveis que são modificadas durante a execução do software. A este processo chamaremos de carregamento e realocação internos, pois o próprio software dedicado será responsável pela sua efetivação.

Como é preciso executar o software deste trabalho no microcomputador de desenvolvimento, principalmente para testá-lo, antes de levá-lo ao microcomputador dedicado deve-se resolver o problema de carregamento e realocação nos dois casos: para o microcomputador de desenvolvimento e para o microcomputador dedicado. A seção 4.3.1 aborda a proposta para realizar o carregamento e realocação dos segmentos de um programa escrito no modo protegido para execução no microcomputador de desenvolvimento. A seção 4.3.2 aborda a proposta para realizar a mesma tarefa para o microcomputador dedicado.

4.3.1. Carregamento/Realocação no Microcomputador de Desenvolvimento

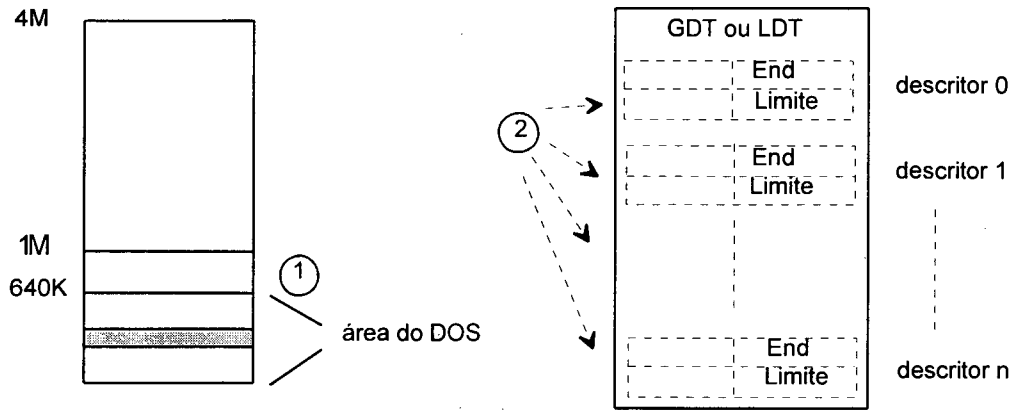
Nesta parte do trabalho o procedimento que será adotado para carregar e realocar o software dedicado no próprio ambiente de desenvolvimento (realocação interna) será descrito. Isto possibilita testar o software antes de sua execução no microcomputador dedicado.

4.3.1.1. Carregamento Embutido no Software de Aplicação

A abordagem a seguir se refere ao software escrito em linguagem assembly. Neste caso, todos os módulos do software de sistema e de aplicação são agrupados num mesmo programa que transfere os segmentos do modo protegido para os endereços desejados. Se os endereços já forem conhecidos antes da transferência, os descritores já podem inclusive contê-los e, portanto, não precisam mais ser alterados. Neste caso os descritores estão **consistentes**. Se os endereços forem determinados somente durante a execução do programa, os descritores correspondentes devem ser alterados usando-se um *alias* para uma das tabelas (segmento de dados mapeado para o mesmo endereço da tabela que contêm o descritor). Neste caso os descritores estão inicialmente **inconsistentes**.

A construção das tabelas faz parte da construção do sistema e deve ser efetuada durante a inicialização (modo real) com o auxílio do **construtor.** As operações de carregamento e realocação devem ser feitas após a passagem para o modo protegido e fazendo uso do **realocador interno**. Tanto o **construtor** quanto o **realocador interno** realizarão suas tarefas com a utilização de macros. **Macros** são trechos de programa em linguagem Assembly que efetuam uma ou mais operações utilizadas com maior frequência. Algumas macros construídas são mostradas no **apêndice A** deste trabalho. A macro n° 4, por exemplo, é responsável pelo carregamento de um trecho ou segmento de programa para uma área de memória qualquer, bastando passar como parâmetros o endereço e o tamanho do trecho ou segmento a ser carregado. O endereço destino pode ser buscado numa posição de memória qualquer ou diretamente na GDT. A macro n° 2 é responsável pela criação ou ajuste de um descritor na GDT. Todas as informações necessárias para a definição de um descritor devem ser passadas como parâmetro para essa macro (ver apêndice A).

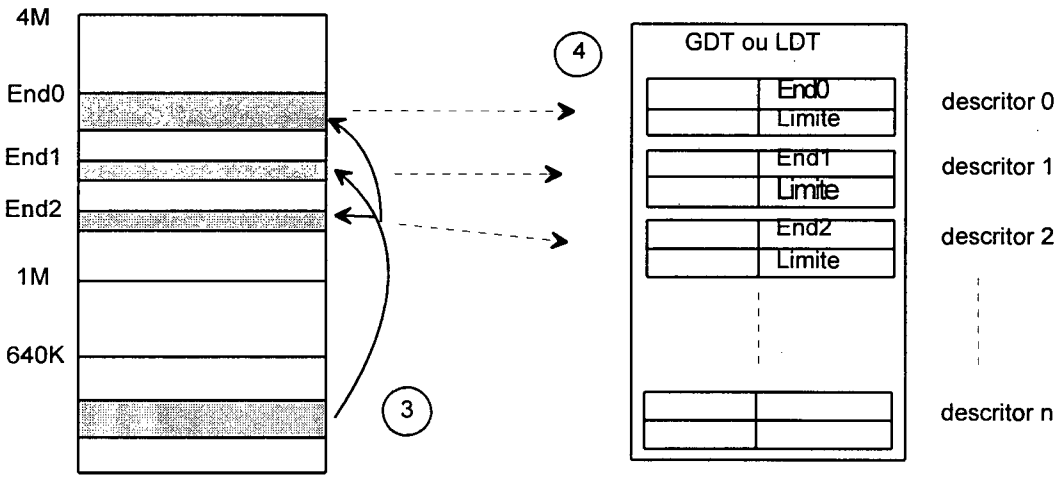
Após efetuado o carregamento e o ajuste no respectivo descritor (caso necessário), o programa pode acessar os dados no segmento correspondente. O procedimento para o caso do descritor **inconsistente** é mostrado na figura 4.2 e para o descritor **consistente** é mostrado na figura 4.3. Os exemplos realocam os segmentos para uma área acima de 1 Mbytes mas é possível realocar para qualquer endereço desejado.



1. Todos os segmentos do software de aplicação e de sistema são carregados na memória do microcomputador de desenvolvimento pelo DOS numa área qualquer abaixo de 640K.

2. Os endereços dos segmentos nos descritores estão inconsistentes, pois o DOS os carregou em endereços aleatórios abaixo de 640K.

(a)

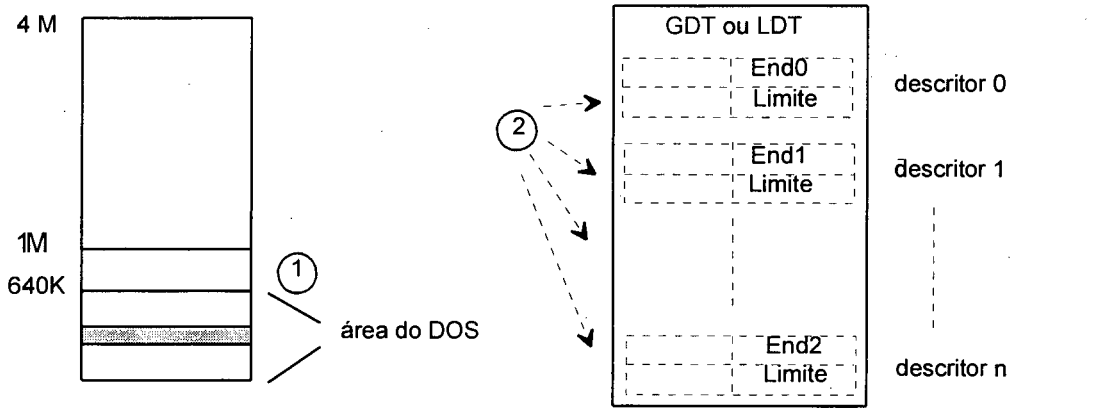


3. Com a utilização do realocador proposto, os segmentos no modo protegido são transferidos para endereços especificados (Ex: End0, End1 e End2).

4. Os descritores são atualizados a fim de manter a coerência com os endereços de carregamento dos segmentos concluindo a realocação.

(b)

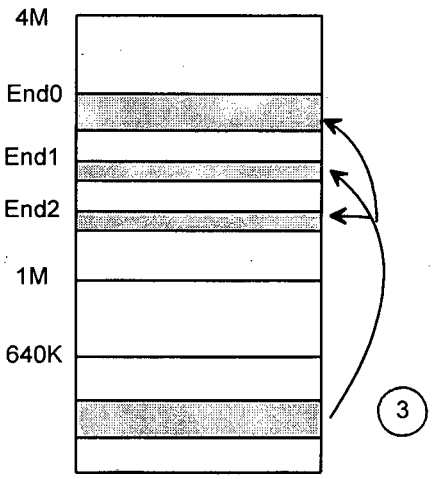
Figura 4.2 Situação do software de sistema e de aplicação no microcomputador de desenvolvimento com os descritores inconsistentes. (a) No momento do carregamento pelo DOS; (b) Após a realocação.



1. Todos os segmentos do software de aplicação e de sistema são carregados na memória do microcomputador de desenvolvimento pelo DOS na área abaixo de 640K.

2. Os endereços dos segmentos nos descritores já possuem os endereços corretos para áreas acima de 1 MBytes. Estes valores são lidos com o auxílio de um segmento alias para a tabela..

(a)



3. Com a utilização do realocador, os segmentos no modo protegido são transferidos para os endereços lidos nos descritores (Ex: End0, End1 e End2) completando a realocação.

(b)

Figura 4.3 Situação do software de sistema e de aplicação no microcomputador de desenvolvimento com os descritores consistentes. (a) No momento do carregamento pelo DOS; (b) Após a realocação.

Para softwares escritos na linguagem C, com o uso do compilador da Microsoft, a realocação é feita de modo semelhante apesar da estrutura de segmentos padrão dos programas. No modelo de programação SMALL, os segmentos de código, dados e pilha devem ter até 64K. O compilador da Microsoft define 9 segmentos para o programa encapsulando-os, a exceção do segmento de código, em um único grupo de segmentos. Esse grupo é chamado DGROUP. Ele é definido para indicar ao ligador que qualquer referência a dados deve ser feita em relação à um único segmento de dados em comum. Isto implica que todos os segmentos de dados devem ser contíguos e limitados a 64k de memória.

Embora os segmentos sejam definidos pelo próprio compilador, pode-se criar descritores com os endereços de trechos de código e dados contidos nos segmentos de código e do DGROUP de forma que se definam outros segmentos para operação no modo protegido. O procedimento, portanto, é o mesmo adotado para os programas em Assembly: carregar um trecho para qualquer área e ajustar o descritor.

4.3.2. Carregamento/Realocação no Microcomputador Dedicado

O carregamento e a realocação para o microcomputador dedicado será feita praticamente da mesma forma descrita para o microcomputador de desenvolvimento (seção 4.2.1.1). Porém, é preciso passar o software dedicado por uma etapa a mais: a realocação externa. A realocação externa terá efeito somente para os segmentos do modo real. Para os segmentos do modo protegido será usado o carregamento e realocação idênticos ao apresentado para o microcomputador de desenvolvimento (realocação interna). Aqui, contudo, os segmentos serão carregados e realocados para a memória RAM.

A realocação externa será feita para definir os endereços dos segmentos que ficarão em EPROM e RAM. Os segmentos alocados em RAM deverão ser carregados (copiados) da EPROM para RAM. Esse é o caso dos segmentos de dados (dos modos real e protegido) e do próprio software de aplicação.

Para os segmentos no modo protegido, os endereços já vão estar definidos nos descritores associados, que podem estar na EPROM ou RAM. A partir daí utiliza-se a ferramenta de realocação proposta para copiar os segmentos do software de aplicação, incluindo os segmentos de dados que contêm as variáveis e eventuais partes do software de sistema, da EPROM para a RAM. Neste caso, não se precisa sequer alterar os descritores, pois sabe-se de antemão em que endereços carregar cada segmento. Feito isto, pode-se chavear para o primeiro segmento executável do software de aplicação.

O procedimento acima é aplicado tanto para o software escrito em assembly quanto em C. Porém, no software em C deve-se alterar o *startup* do compilador para adaptá-lo ao microcomputador dedicado. O *startup* é um programa em assembly que é acoplado a todo programa desenvolvido pelo compilador da Microsoft que realiza uma série de operações de inicialização. O próprio *startup* inclusive pode realizar a tarefa de copiar os dados da EPROM para RAM.

Uma ferramenta que pode ser utilizada para testar o software no microcomputador dedicado antes de gravar a EPROM é o simulador de EPROM. A seção seguinte fala do simulador de EPROM e como utilizá-lo neste trabalho.

4.3.2.1. Simulador de EPROM

Uma das tarefas mais tediosas no desenvolvimento de software para microcomputadores dedicados é o ciclo de gravação e apagamento da EPROM, onde os códigos objetos são gravados no *chip* de uma EPROM, inserida na placa do sistema. Se algum erro for detectado, o que na maioria das vezes acontece, deve-se extrair o *chip*, expô-lo a luz ultravioleta e reprogramá-lo com o software depurado.

O simulador de EPROM é uma ferramenta que consiste, basicamente, de memória RAM e software. É conectado no soquete da EPROM e simula o seu funcionamento [MIP90]. Isto permite carregar em sua RAM o código que executaria na EPROM quantas vezes for necessário até que o software esteja livre de erros (*bugs*). A figura 4.4 descreve

a operação do simulador de EPROM. Esta ferramenta possibilita o teste de todo o sistema sem gravar sequer um byte em EPROM. Uma vez detectado algum erro, o desenvolvimento é retomado ao microcomputador de desenvolvimento pelas etapas de edição, compilação e ligação e posterior envio de volta para o simulador para nova conferência. Este ciclo é repetido até que se esteja satisfeito com o software quando então se pode gravar a EPROM apenas uma vez.

Utilizando o simulador de EPROM as eliminações de erros são mais simples e práticas. O software a ser testado no simulador é exatamente o mesmo que irá ser executado no microcomputador dedicado, inclusive utilizando os mesmos endereços, caso tivesse sido gravado em EPROM. Portanto, nenhuma modificação é necessária com relação ao carrregamento e realocação de segmentos com a utilização do simulador.

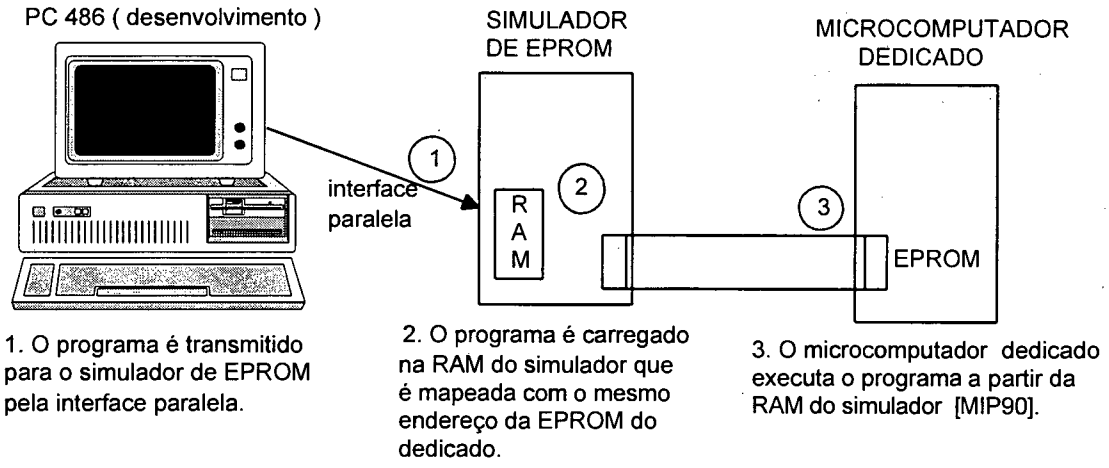


Figura 4.4 - Conexão micro de desenvolvimento - simulador de EPROM - microcomputador dedicado.

5. Implementação das Ferramentas

5.1. Introdução

Este capítulo descreve a implementação das ferramentas definidas no capítulo 3. Cinco grupos de ferramentas foram implementadas: **construtor**, **realocador interno** e **externo**, **carregador**, **mini-DOS** e **depurador**. A seguir será feita uma descrição da implementação de cada ferramenta.

5.2. O Construtor

A ferramenta **construtor** foi implementada em linguagem Assembly sob a forma de macros. A linguagem Assembly é a única linguagem de programação que permite manipular-se diretamente com os registradores especiais da arquitetura do 80386. Como a construção do sistema envolve a manipulação dos registradores das tabelas de sistema, as macros foram implementadas com a utilização do Assembly. As macros são uma forma de facilitar a execução de tarefas em um programa, pois o programador precisa apenas conhecer como passar os parâmetros necessários para que as mesmas cumpram suas tarefas. A tabela 5.1 mostra as macros implementadas para o **construtor**.

Macro	Descrição
Gera_Desc	Aloca um descritor na GDT com os dados passados.
Gera_Gate	Aloca um *gate* na IDT com os dados passados.
Gera_TSS	Aloca um TSS com os dados passados.
Monta_GDT	Monta a GDT com o número de descritores passados.
Monta_IDT	Monta a IDT com o número de *gates* passados.

Tabela 5.1 - Macros que integram o **construtor**.

5.3. O Realocador

A ferramenta para a realocação de segmentos foi implementada em duas partes distintas: o **realocador interno** e o **realocador externo**. A seguir é feita uma descrição destas duas ferramentas.

5.3.1. O Realocador Interno

O **realocador interno** foi implementado em linguagem Assembly sob a forma de macros. O **realocador interno** tem a função de manter coerente o endereço de carregamento de um segmento com o endereço base presente em seu descritor. A tabela 5.2 mostra as macros implementadas para o **realocador interno**.

Macro	Descrição
Copia_Seg_RAM	Copia um segmento ou trecho para uma área de memória qualquer (ex: de ROM para RAM).
Copia_Tab_RAM	Copia uma tabela (GDT, LDT ou IDT) para uma área de memória qualquer (ex: de ROM para RAM).

Tabela 5.2 - Macros que integram o **realocador interno.**

5.3.2 - O Realocador Externo

O **realocador externo** foi implementado em linguagem C sob a forma de um programa. Esta ferramenta tem a função de resolver as referências aos segmentos do modo real, as quais, os compiladores não podem efetuar por não conhecerem onde os mesmos serão carregados. Estes valores podem se referir tanto à memória EPROM quanto à RAM e devem ser especificados no momento da realocação.

Para efetuar sua tarefa é preciso que o **realocador externo** identifique quais segmentos realocar para endereços em RAM e EPROM e quais endereços utilizar para cada tipo de memória. No realocador implementado, é preciso que o aplicativo especifique a classe de cada segmento do modo real declarado, conforme sua localização desejada: ROM ou CODE para segmentos em EPROM, e RAM para segmentos em RAM. Em Assembly a declaração dos segmentos é feita pelo próprio aplicativo, assim sendo não há problemas para atribuir-lhes a classe desejada. Porém, na linguagem C algumas considerações devem ser notadas. A seção seguinte aborda a realocação externa para aplicativos implementados em C.

Com relação aos endereços de realocação, o próprio realocador especifica o endereço F0000 para EPROM, que é o padrão de uma placa baseada nos processadores x86. O endereço de realocação para RAM é definido no momento da realocação. Assim sendo, o realocador deverá receber como entrada o arquivo executável e o arquivo de mapeamento de segmentos (.MAP) que contém a lista dos segmentos do aplicativo juntamente com a classe de cada um. De posse destas informações, a realocação pode ser realizada. A figura 5.1 mostra o **realocador externo** e suas entradas e saída.

Outra função do realocador externo implementado é definir o endereço de *bootstrap,* que é o endereço onde inicia a execução do programa. Este endereço fica mapeado para memória ROM e é definido pelo próprio 80386 como sendo FFFF0 (1M -16 bytes). O realocador coloca nesta posição o código de um desvio incondicional para o ponto de entrada do programa - a inicialização.

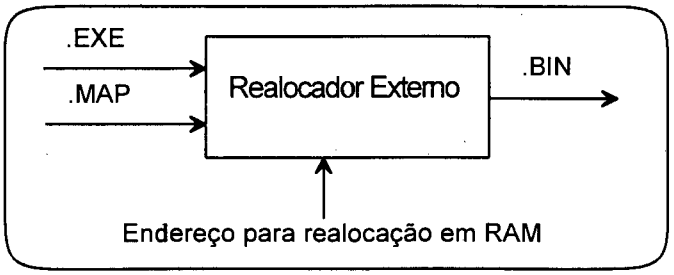


Figura 5.1 - O **realocador externo** e suas entradas e saída.

5.3.2.1. Realocação Externa para Linguagem C

Na linguagem C, ou em outra linguagem de alto nível, o aplicativo não declara segmentos dentro do programa. Esta tarefa é realizada por um módulo responsável pela preparação do ambiente de execução do aplicativo. No caso da linguagem C, os compiladores acoplam ao arquivo executável um módulo que executa uma série de inicializações, dentre elas a declaração de segmentos. No compilador Microsoft C este módulo chama-se *startup.*

O *startup* foi desenvolvido, entretanto, para definir o ambiente de execução do aplicativo sob o sistema operacional MS-DOS e portanto, deve ser alterado para o ambiente dedicado. Na verdade, a alteração é mais no sentido de retirar procedimentos desnecessários e incompatíveis com o ambiente protegido do dedicado. Os parâmetros de chamada da função *main* ( ), por exemplo, devem ser nulos, pois não há linha de comando no ambiente dedicado.

Quanto aos segmentos do programa, estes são declarados no *startup* e obedecem um padrão pré-definido. No modelo de memória SMALL, por exemplo, são definidos um segmento para código e um grupo de segmento (DGROUP) que engloba dados, variáveis inicializadas, variáveis não-inicializadas, constantes e pilha. Antes de chamar a função *main* ( ), o *startup* carrega os registradores DS, ES e SS com o endereço de DGROUP.

Portanto, é preciso incluir na declaração dos segmentos o atributo de classe ROM para o segmento de código, e RAM para o DGROUP.

5.4. O Carregador

Existem duas formas de carregar os aplicativos no microcomputador dedicado. A primeira é realizada pelo próprio software de sistema no momento da inicialização através da utilização conjunta das ferramentas **construtor** e **realocador interno** — descritas nas seções 5.2 e 5.3. A segunda alternativa, objeto da ferramenta **carregador**,

permite que aplicativos sejam carregados pela serial e executados no dedicado através de comandos solicitado pelo terminal. Os comandos do carregador são descritos na seção 5.4.2. Esta ferramenta pode explorada como recurso para testar e depurar os aplicativos antes de incluí-los na EPROM do dedicado.

O **carregador** foi implementado em Assembly, e exigiu a definição de um método especial para o desenvolvimento de aplicativos que fossem direcionados para este tipo de carregamento. O método definido é apresentado a seguir.

5.4.1. Método de Desenvolvimento de Aplicativos para Carregamento Via Serial

A figura 5.2 mostra o método definido para o desenvolvimento de aplicativos para este tipo de carregamento. Duas etapas apenas são necessárias: compilação e ligação, utilizando ferramentas para o ambiente MS-DOS, e conversão do arquivo executável (.EXE) para um arquivo binário (.BIN).

Figura 5.2 - Etapas para o desenvolvimento de aplicativos para carregamento via serial.

Uma observação importante é com relação à etapa de conversão para binário. O arquivo executável gerado pelas ferramentas não possui nenhuma referência a segmentos realocáveis, pois o programa é desenvolvido puramente no modo protegido. Por esse motivo, o arquivo não precisa passar por um realocador externo. Apenas o cabeçalho, presente em qualquer executável deve ser retirado para efetuar o seu carregamento pela serial. Isso é feito utilizando um programa conversor para binário. O utilitário do MS-DOS que converte programas executáveis de extensão ".EXE" para ".COM" (EXE2BIN) não pode ser utilizado porque ele só suporta programas com apenas um

segmento, e no procedimento definido não há essa limitação, podendo-se desenvolver aplicativos com o número de segmentos desejados.

Existe a necessidade, entretanto, de que o **carregador** no dedicado tenha as informações necessárias para que ele possa efetuar sua tarefa corretamente. Informações como o número de segmentos e tamanho de cada segmento do aplicativo devem ser enviadas juntamente com o arquivo. Devido a isto, foi criado um tipo de cabeçalho que contém as seguintes informações: nome do aplicativo, número de segmentos existentes e o descritor de cada um, contendo o endereço de carregamento, tamanho e direitos de acesso. Isto permite que o próprio programa defina os endereços para cada segmento. O cabeçalho proposto é incluído no primeiro segmento do programa criado especialmente para esse fim, e não é incluído no número de segmentos do aplicativo. A figura 5.3 mostra um exemplo de cabeçalho para um aplicativo a ser carregado pela serial.

```
EXTRA SEGMENT

nome      DB 'Embed$'
num_seg   DB 2

          DW  Tam_Dados
          DW  End_Base_Dados_0_15
          DB  End_Base_Dados_16_.23
          DB  Acesso_Dados
          DB  Granularidade_Dados
          DB  End_Base_Dados_24_31

          DW  Tam_Código
          DW  End_Base_Código_0_15
          DB  End_Base_Código_16_23
          DB  Acesso_Código
          DB  Granularidade_Código
          DB  End_Base_Código_24_31

EXTRA ENDS
```

Figura 5.3 Exemplo de cabeçalho para aplicativos carregados pela serial.

5.4.2. Comandos do Carregador

Os comandos do carregador são:

RECEIVE (R)

Esse comando põe o MONITOR no estado de espera de um arquivo pela interface serial. Cada arquivo constitue um aplicativo escrito no modo protegido que é enviado do microcomputador emulando um terminal para o dedicado.

O primeiro segmento recebido contém os descritores dos segmentos do aplicativo com todas as informações necessárias para o carregamento e a realocação dos mesmos. De posse dos endereços de base e tamanho contidos nos descritores, cada segmento é recebido e carregado na memória e o seu descritor alocado na GDT. Ao final do carregamento e realocação o MONITOR devolve ao terminal o seletor atribuído ao segmento de código inicial do aplicativo, que adotou-se como sendo o último segmento recebido.

Forma de usar: R

LISTA_APLICATIVOS (L)

Lista o nome e o seletor de cada software aplicativo carregado na memória do dedicado seja pela interface serial ou pelo módulo de inicialização INIC. Todos os aplicativos listados por esse comando estão disponíveis para execução.

Forma de usar: L

GO (G)

Executa um programa aplicativo carregado no dedicado. Após o término da execução, o controle é passado de volta ao MONITOR.

Se alguma instrução for executada em um endereço que contém um dos *breakpoints* ativos, a exceção de depuração é gerada e o programa entra no estado de execução passo-a-passo.

Forma de usar: G *seletor*

onde *seletor* é o seletor de código inicial de um programa aplicativo.

HELP (H)

Exibe a lista dos comandos disponíveis.

Forma de usar: H

5.5. O Mini-DOS

O **Mini-DOS** foi implementado através da adaptação das funções mais comuns da INT 21h do MS-DOS de forma que elas pudessem ser utilizadas pelo modo protegido. As funções implementadas foram basicamente aquelas referentes a entrada e saída, porém todas foram redirecionadas do teclado e vídeo para a interface serial. A tabela 5.3 mostra as funções da INT 21h implementadas.

Número	Função
01h	Entrada de caractere com eco.
02h	Saída de caractere.
06h	Entrada e saída direto do console.
08h	Entrada de caractere sem eco.
09h	Apresentação de *string*.
30h	Obtém o número da versão do Mini-DOS.

Tabela 5.3 - Funções do **Mini-DOS** implementadas.

5.6. O Depurador

O **depurador** foi implementado a fim de possibilitar a depuração de aplicativos através da utilização dos recursos da arquitetura do 80386. Esta ferramenta pode ser utilizada pelo projetista do sistema para depurar, por exemplo, os aplicativos carregados pela serial através do **carregador**. O **depurador** também utiliza a serial para receber os comandos de depuração do terminal. A interface com o usuário se restringe a um *prompt* por onde os comandos, representados por letras, são esperados. Os comandos do **depurador** são apresentados a seguir.

5.6.1. Comandos do Depurador

DUMP (D)

Executa um *dump* na memória do dedicado mostrando o conteúdo em hexadecimal e ASCII.

Forma de usar: D *end1*,*end2*

onde *end1* é o endereço de 32 bits do início do *dump*.

end2 é o endereço de 32 bits do final do *dump*.

SETA_BREAKPOINT (B)

Define um *breakpoint* na memória. Até 4 *breakpoints* podem ser definidos em memória RAM ou ROM utilizando os recursos da arquitetura do 80386.

Forma de usar: B *seletor*,*offset*,*num*

onde *seletor* é o seletor do segmento destino do *breakpoint*.

offset é o offset em relação ao segmento destino do *breakpoint*.

num é o número do *breakpoint* (0 ~ 3).

RESETA_BREAKPOINT (C)

Retira o *breakpoint* da memória do dedicado.

Forma de usar: C *num*

onde: *num* é o número do *breakpoint* a ser retirado da memória.

LISTA_BREAKPOINTS (I)

Lista os *breakpoints* ativos juntamente com o seletor:offset de localização.

Forma de usar: I

GO_STEP (S)

Executa um programa passo-a-passo. A cada instrução executada a exceção de depuração é gerada. O conteúdo de todos os registradores da CPU é enviado para o dedicado, assim como os próximos 10 bytes de instruções. A qualquer instante é possível interromper a execução passo-a-passo e retornar a execução normal do programa.

Forma de usar: S *seletor*

onde: *seletor* é o seletor do segmento inicial de execução de um programa.

5.7. Interrupções e Exceções

A fim de viabilizar a utilização das ferramentas descritas, foram implementadas rotinas de serviço para as seguintes interrupções e exceções:

5.7.1. Interrupções

Foram implementadas as seguintes rotinas de serviço de interrupção:

Número da Interrupção	Função
03h	Interrupção de *breakpoint*.
70h	Interrupção do relógio.
71h	Interrupção do teclado.
84h	Interrupção da interface de comunicação (serial). AH=0 Inicializa porta de comunicação. AH=1 Escreve um caractere na porta de comunicação. AH=2 Lê um caractere da porta de comunicação. AH=3 Obtém o status da porta de comunicação.

Tabela 5.4 - Interrupções implementadas.

5.7.2. Exceções

As seguintes exceções foram implementadas:

Número da Exceção	Tipo
01h	Exceção de depuração.
0Ch (12d)	Exceção de falha na pilha.
0Dh (13d)	Exceção de falha de proteção geral.

Tabela 5.5 - Exceções implementadas.

5.8. Conclusão

A implementação das ferramentas foi feita visando sua utilização num ambiente dedicado a fim de auxiliar no desenvolvimento dos softwares de sistema e de aplicativos. A linguagem Assembly foi utilizada para a implementação das ferramentas, com exceção do carregador que foi implementado em C.

Da forma como as ferramentas foram implementadas, é possível utilizá-las somente no desenvolvimento de software dedicado em linguagem Assembly. No caso do desenvolvimento ser realizado com a linguagem C, é preciso adaptá-las para os padrões de segmentação e de chamada de função específicos desta linguagem.

6. Utilização das Ferramentas na Simulação de um Ambiente Dedicado

6.1. Introdução

Este capítulo descreve a utilização das ferramentas implementadas na simulação de um ambiente dedicado. Um PC 386 compatível foi a máquina utilizada para esta simulação.

Para a simulação, foram utilizados dois aplicativos simples, que serviram como exemplo de utilização das ferramentas implementadas. Um aplicativo, APL1, foi carregado no ambiente simulado, através da utilização do **construtor** e **realocador interno**. O segundo aplicativo, denominado EMBED1, foi carregado pelo **carregador** tendo sido desenvolvido conforme o método apresentado para o carregamento via serial. Os dois aplicativos exemplos eram formados por apenas três segmentos: código, dados e pilha e se restringiam ao envio de mensagens para o terminal.

A simulação foi realizada através da utilização das ferramentas e da execução dos aplicativos de forma independente do sistema operacional e da BIOS da máquina, que são usados apenas para carregar o software na memória. Este procedimento pode ser utilizado naturalmente para o desenvolvimento e teste de aplicativos, já que os ambientes simulado e dedicado são similares, só devendo diferir com relação às entradas e saídas, que podem ser específicas no dedicado. Ao final do capítulo será mostrada a forma de portar o ambiente simulado para a própria EPROM do dedicado e as implicações decorrentes.

6.2. Configuração do PC Simulando o Dedicado

A figura 6.1 mostra a configuração geral da simulação do ambiente dedicado. O desenvolvimento do software dedicado utilizou as ferramentas DOS para a compilação e

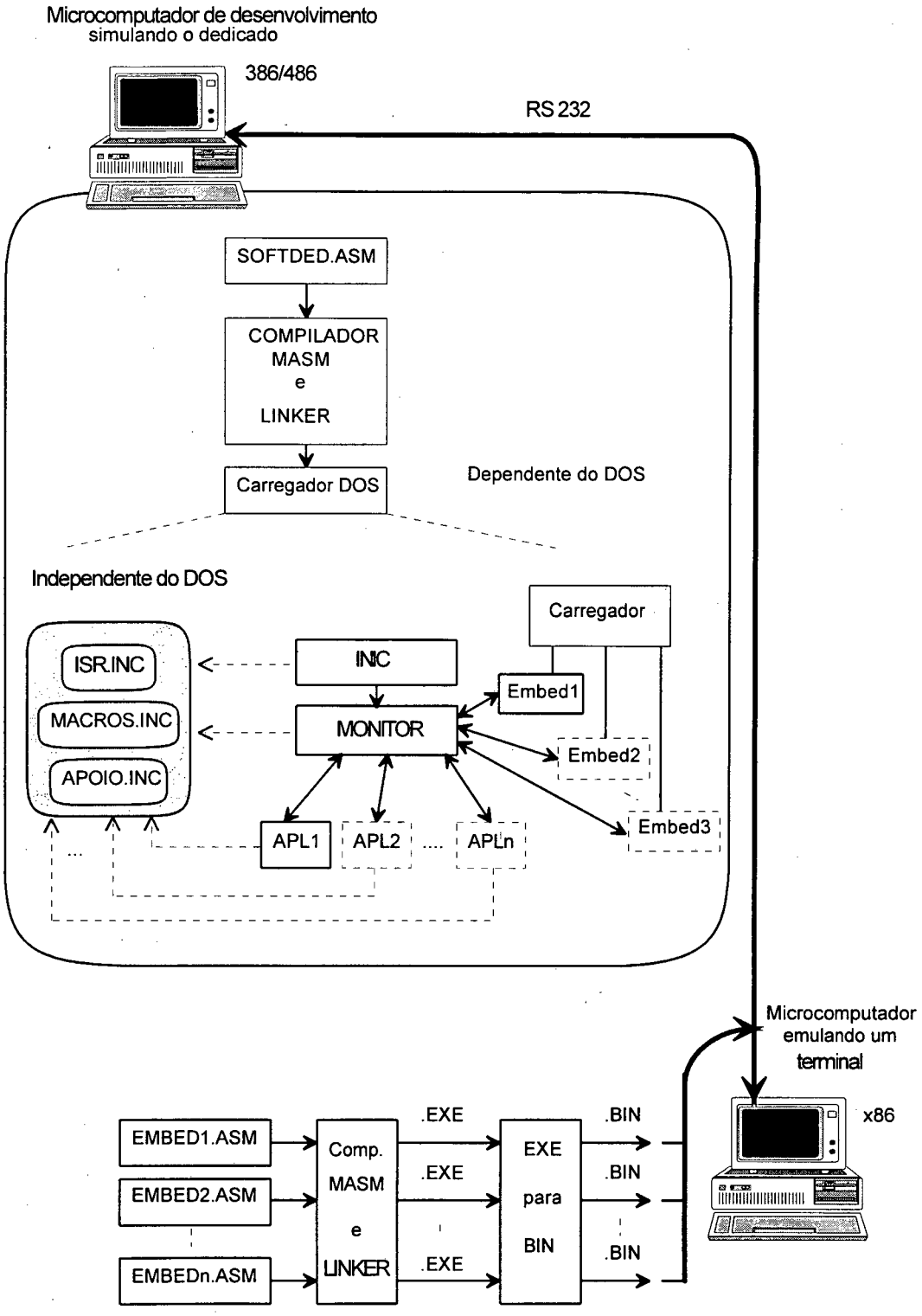


Figura 6.1 - Configuração da simulação do ambiente dedicado.

ligação. O compilador MASM (Microsoft Macro Assembler) e o LINKER foram utilizados para executar esta tarefa.

O carregamento do software na memória do PC é feito usando o carregador — loader — do MS-DOS, que passa o controle para o módulo de inicialização — INIC. Essa é a última tarefa dependente do MS-DOS na simulação do ambiente dedicado. A partir deste ponto os passos são realizados totalmente independentes do DOS e da BIOS da máquina.

6.2.1. O Módulo INIC

O módulo INIC é responsável pela inicialização do hardware e pela preparação do ambiente de simulação do dedicado. Ele possui a primeira parte escrita no modo real, que é o modo inicial do processador, e uma segunda escrita após a passagem para o modo protegido. As diversas etapas realizadas pelo módulo INIC são mostradas na figura 6.2 e apresentadas a seguir.

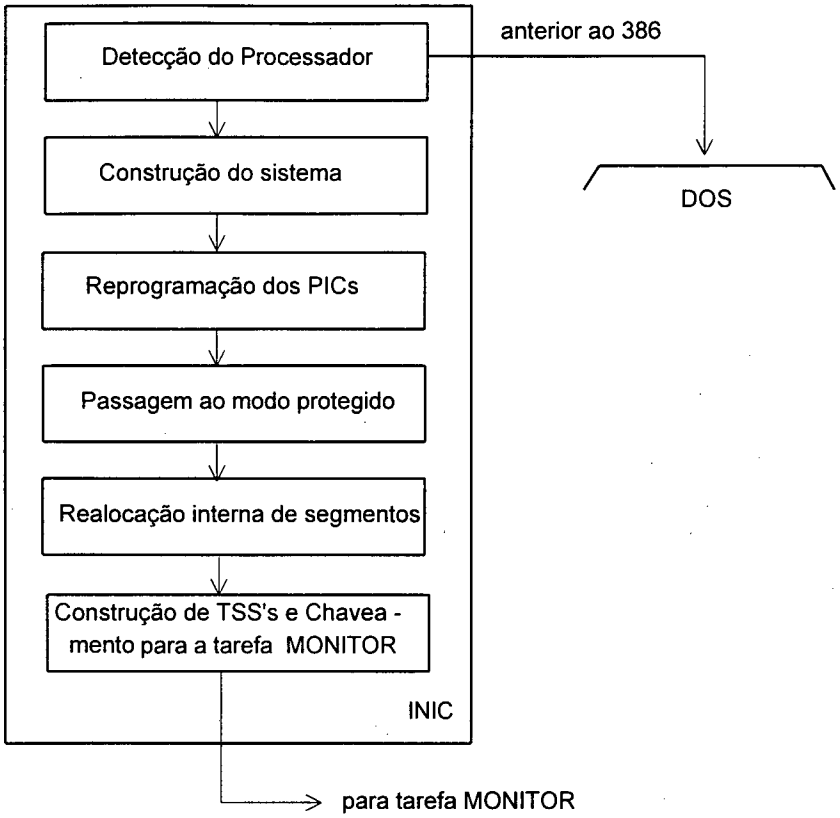


Figura 6.2 - Etapas do módulo INIC.

6.2.1.1. Detecção do Processador

O primeiro passo do módulo INIC é determinar qual processador é utilizado como CPU do PC. O programa foi desenvolvido utilizando os recursos do 386 em diante, sendo assim, sua execução num processador antecessor leva a consequências imprevisíveis. Dessa forma se for detectada a presença de um processador anterior ao 80386, uma mensagem é enviada à tela e o programa devolve o controle para o MS-DOS. Essa é a única utilização de funções do MS-DOS pelo programa. Após detectada a presença de um 80386 ou superior, o **Mini-DOS** é instalado com as funções mais usuais do MS-DOS.

6.2.1.2. Construção do Sistema

Antes do módulo INIC realizar a passagem para o modo protegido, é preciso montar as tabelas do sistema. A tabela de descritores globais (GDT) e a tabela de descritores de interrupção (IDT) foram montadas com o auxílio das macros instaladas com o arquivo MACROS.INC descrito posteriormente.

Foram alocados na GDT, descritores para os segmentos dos módulos INIC e MONITOR (código, dados, pilha e TSS), bem como para o aplicativo exemplo - APL1 carregado juntamente com o software. Além destes, foi reservado um número de descritores para os aplicativos carregados pela interface serial. Na IDT foram alocados *gates* para todas as rotinas de serviço de interrupção ou exceção implementadas neste trabalho. A figura 6.3 mostra um esboço das tabelas após a construção do sistema.

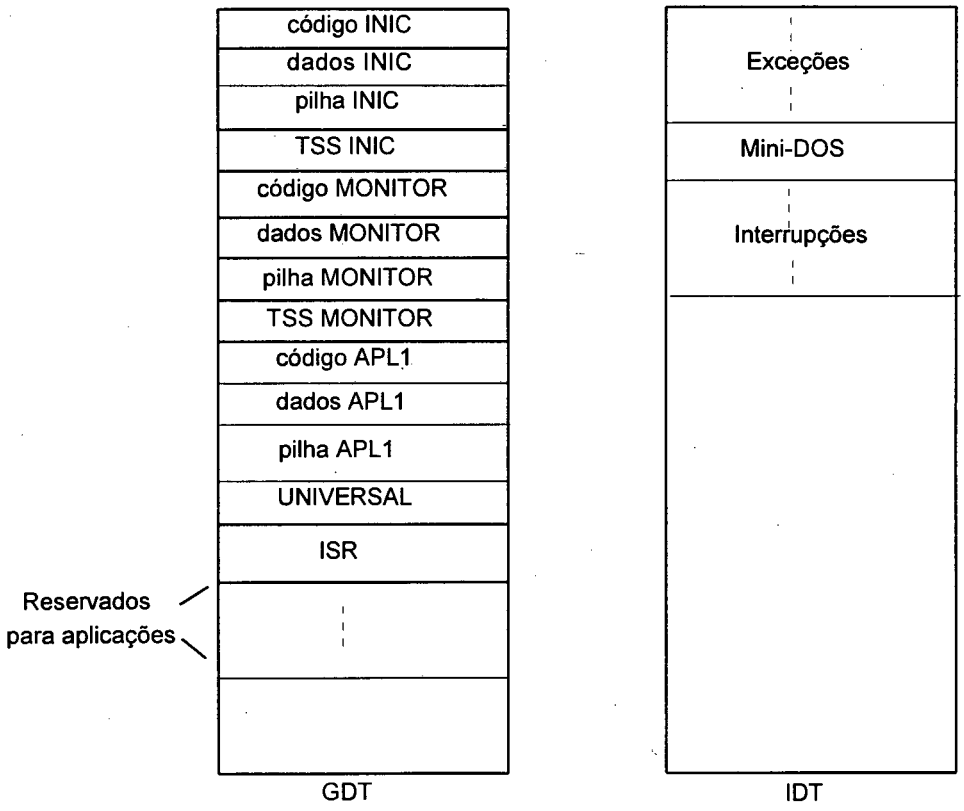


Figura 6.3 - Esboço das tabelas GDT e IDT após a montagem.

Para montar descritores é preciso basicamente determinar o endereço base de 32 bits, o tamanho e os direitos de acesso de cada segmento do programa. O endereço de base é o endereço linear da primeira posição onde o segmento será acessado na memória, portanto, é indispensável que esses valores estejam sempre de acordo com os endereços onde os segmentos estão armazenados. Semelhantemente, para montar *gates* é preciso determinar o endereço base de 32 bits de cada rotina de serviço de interrupção do programa.

Como o MS-DOS carrega os segmentos de forma imprevisível nos primeiros 640K bytes da memória do PC, foi necessário descobrir estes endereços para montar os descritores e *gates*. Tanto a linguagem Assembly, que possui segmentos definidos pelo próprio programador, quanto a linguagem C, que possui uma estrutura de segmentos pré-definida, dispõem de recursos para determinar o endereço de cada segmento do programa, não constituindo problema para esta tarefa.

O tamanho dos segmentos também não é problema para as linguagens que possuem artifícios para determiná-los.

Os direitos de acesso estão especificados da seguinte forma: todos os descritores possuem *DPL*=0, ou seja, estabelecem que os segmentos executem com o maior nível de privilégio, *P*=1 indicando segmentos presentes na memória e *tipo* conforme suas características (código somente para execução, código para execução e leitura, dados somente para leitura, dados para leitura e escrita). Qualquer violação dos direitos de acesso causa uma exceção de falha de proteção geral (exceção número 13).

Na IDT optou-se por alocar *gates* de *trap* em vez de *gates* de interrupção. Na ocorrência de uma interrupção, os *gates* de *trap* não modificam o *flag* de interrupção do processador, enquanto que os *gates* de interrupção desabilitam as interrupções.

6.2.1.3. Reprogramação dos PICs

A arquitetura PC-AT possui 2 controladores de interrupção programáveis, responsáveis pelo gerenciamento das interrupções de hardware da placa. Cada PIC possui 8 entradas de interrupção (IRQ0 até IRQ7 no primeiro PIC e IRQ8 até IRQ15 no segundo) que podem ser conectadas com as linhas de solicitação de interrupção de até 8 dispositivos diferentes. A linha IRQ2 do primeiro PIC serve como entrada para as solicitações do segundo PIC. As interrupções são priorizáveis sendo a IRQ0 a de maior prioridade e a IRQ7 a de menor prioridade. Quando uma interrupção ocorre, o PIC envia para a CPU um vetor que identifica o seu número para que seja executada a rotina de serviço (ISR). Se duas interrupções ocorrerem ao mesmo tempo o PIC selecionará a de maior prioridade.

Dentre as 256 interrupções possíveis (INT0 ~ INT255) dos processadores x86, os PICs são originalmente programados para enviar vetores correspondentes às INT8 ~ INT0F (PIC1) e INT70 ~ INT77 (PIC2). Entretanto, o modo protegido não é compatível com essa configuração, reservando para o sistema e futuras implementações da Intel, as interrupções de número 0 até a 31 (INT0 ~INT31). As interrupções de número 0 a 15 são especialmente dedicadas ao tratamento das exceções. Por esse motivo, efetuou-se a reprogramação dos PICs para enviar vetores correspondentes às interrupções de número 70 ~77 (PIC1) e 78 ~ 7F (PIC2).

6.2.1.4. Passagem ao Modo Protegido

Uma vez montadas a GDT e IDT, o módulo INIC realiza a passagem ao modo protegido e o esvaziamento da fila de instrução QUEUE, a fim de garantir que as instruções sejam lidas a partir de então de acordo com o modo protegido.

6.2.1.5. Realocação de Segmentos

Esta etapa é responsável pelo carregamento e realocação dos segmentos do modo protegido para qualquer área de memória desejada.

Na simulação do ambiente dedicado, todos os segmentos são inicialmente carregados pelo DOS para a memória RAM, mas no dedicado é necessário copiar os segmentos de dados — com tipo de acesso para leitura e escrita —, as tabelas GDT e IDT e os segmentos de estado de tarefa — TSSs — da memória ROM para RAM. Os segmentos de código das aplicações também podem ser copiados para a ROM afim de melhorar o desempenho. Essa tarefa é realizada pelo **realocador interno** presente no arquivo MACROS.INC, que recebe o endereço do segmento ou de uma posição dentro de um segmento, o tamanho e o endereço de destino. No programa optou-se por já definir nos descritores dos segmentos a serem realocados, o endereço base de destino, dispensando suas alterações após a cópia, já que os endereços de execução foram definidos na fase de projeto e são considerados *endereços absolutos*, ou seja, não mudam a cada execução.

No PC, o carregamento e realocação foram simulados usando as mesmas macros para carregar a GDT, IDT, TSS e outros segmentos para endereços acima de 2M bytes. A figura 6.4 mostra um exemplo realocando a GDT para o endereço 2M (200000h).

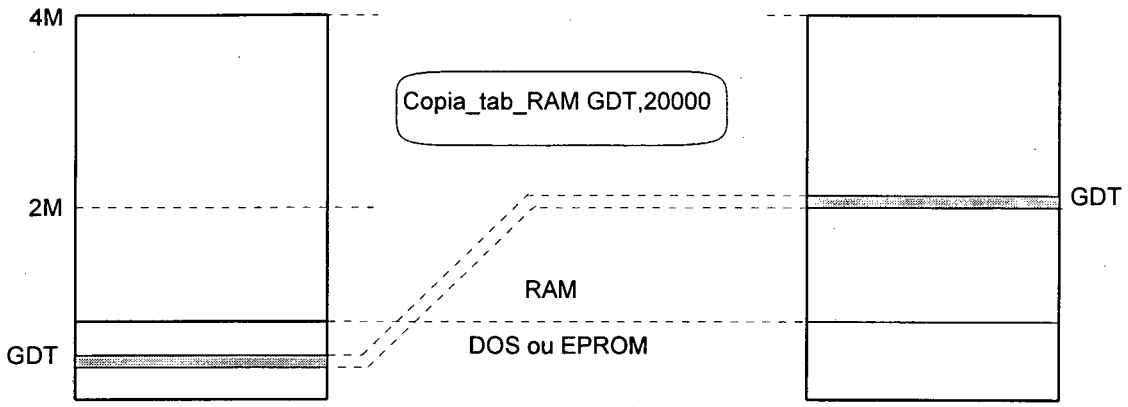


Figura 6.4 - Exemplo de realocação utilizando a GDT.

6.2.1.6. Construção de TSS's e Chaveamento para a Tarefa "Monitor"

Nesta etapa foi construído um TSS para a tarefa "Monitor" que engloba os segmentos do módulo MONITOR. A construção de um TSS é realizada também com o auxílio de uma macro do **construtor** contida no arquivo MACROS.INC e engloba a inicialização de um TSS na memória RAM com os valores passados pela macro. Não foi necessário inicializar um TSS para a tarefa corrente porque o seu estado será automaticamente guardado no TSS no momento do próximo chaveamento. Apenas é necessário inicializar TSS's de tarefas que ainda não executaram pela primeira vez, a fim de estabelecer os estados iniciais das mesmas.

A construção de TSS's deve ser feita após a construção dos descritores de TSS correspondentes, para que a macro possa obter destes o endereço de armazenamento permitindo inicializá-los.

Após a construção dos TSSs é realizado um chaveamento de tarefas. A partir desse momento o módulo INIC passa o controle para o módulo MONITOR e não volta mais a assumir o controle da CPU. A figura 6.5 mostra este chaveamento.

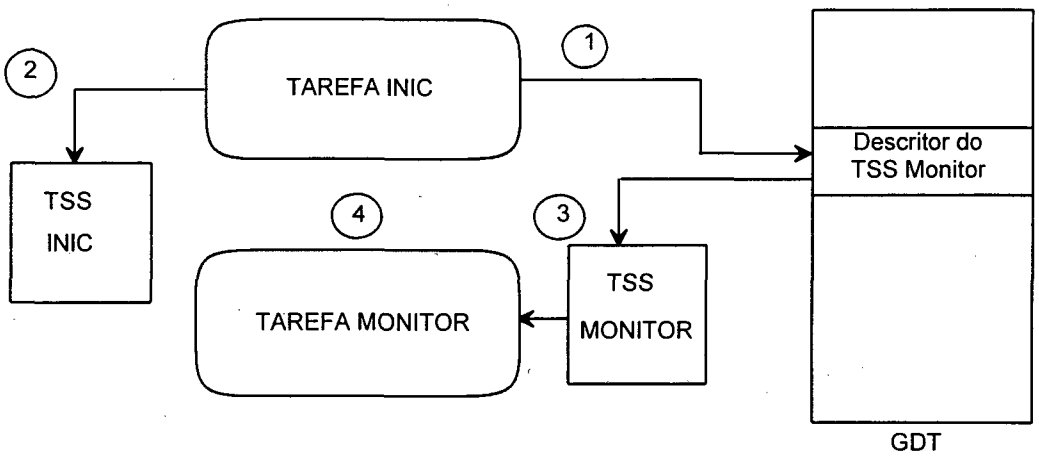


Figura 6.5 - Etapas do chaveamento da tarefa INIC para a tarefa MONITOR.

As etapas do chaveamento apresentadas na figura 6.5 são:

1. A tarefa INIC causa um desvio para o descritor de TSS da tarefa MONITOR;
2. O estado da tarefa INIC é guardado no TSS INIC;
3. O estado inicial da tarefa MONITOR é lido do TSS MONITOR;
4. A tarefa MONITOR recebe o controle da CPU.

6.2.1.7. Exemplo de Utilização das Ferramentas Aplicado à Inicialização do Ambiente Simulado

Nesta seção será resumida a fase de inicialização responsável pela preparação do ambiente de simulação do dedicado. A utilização das ferramentas será demonstrada de forma simples e objetiva sem a preocupação com muitos detalhes.

O ambiente dedicado simulado possui a seguinte estrutura (os endereços utilizados não tem significância neste exemplo) :

Segmentos no modo protegido	**Endereços para carregamento e execução**
Código tarefa INIC (Cod_INIC)	End_Cod_INIC
Dados tarefa INIC (Dad_INIC)	End_Dad_INIC
Pilha tarefa INIC (Pilha_INIC)	End_Pilha_INIC
TSS INIC (TSS_INIC)	End_TSS_INIC
Código tarefa MONITOR (Cod_MON)	End_Cod_MONITOR
Dados tarefa MONITOR (Dad_MON)	End_Dad_MONITOR
Pilha tarefa MONITOR (Pilha_MON)	End_Pilha_MONITOR
TSS MONITOR (TSS_MON)	End_TSS_MONITOR
Código aplicativo APL1 (Cod_APL1)	End_Cod_APL1
Dados aplicativo APL1 (Dad_APL1)	End_Dad_APL1
Pilha aplicativo APL1 (Pilha_APL1)	End_Pilha_APL1

Tabela 6.1 - Estrutura de segmentos exemplo para o ambiente dedicado.

Uma vez definidos os endereços de cada segmento no ambiente dedicado, pode-se utilizar as ferramentas para efetuar a inicialização do mesmo. A seguir será exemplificada a inicialização.

```
Monta GDT n            ; Monta GDT com n descritores
Gera_Desc Cod_INIC     ; Aloca um descritor para código da tarefa INIC
Gera_Desc Dad_INIC     ; Aloca um descritor para  dados da tarefa INIC
Gera_Desc Pilha_INIC   ; Aloca um descritor para pilha da tarefa INIC
Gera_Desc TSS_INIC     ; Aloca um descitor para TSS da tarefa INIC
```

```
Gera_Desc Cod_MON      ; Aloca um descritor para  código da tarefa MONITOR
Gera_Desc Dad_MON      ; Aloca um descritor para  dados da tarefa MONITOR
Gera_Desc Pilha_MON    ; Aloca um descritor para pilha da tarefa MONITOR
Gera_Desc TSS_MON      ; Aloca um descritor para TSS da tarefa MONITOR
Gera_Desc Cod_APL1     ; Aloca um descritor para código do aplicativo APL1
Gera_Desc Dad_APL1     ; Aloca um descritor para dados do aplicativo APL1
Gera_Desc Pilha_APL1   ; Aloca um descritor para pilha do aplicativo APL1
Passa_protegido*       ; Passa para o modo protegido
Gera_TSS  TSS_INIC     ; Monta o TSS para tarefa INIC
Gera_TSS  TSS_MON      ; Monta o TSS para tarefa MONITOR
Copia_Seg_RAM  Cod_APL1, End_Cod_APL1      ;Carrega Cod_APL1 para
                                           End_Cod_APL1
Copia_Seg_RAM  Dad_APL1, End_Dad_APL1      ;Carrega Dad_APL1 para
                                           End_Dad_APL1
Copia_Seg_RAM  Pilha_APL1, End_Pilha_APL1  ;Carrega Pilha_APL1 para
                                           End_Pilha_APL1
Copia_Tab_RAM  GDT, End_GDT                ;Carrega GDT para
                                           End_GDT
Chaveia TSS_MON*       ; Chaveia para a tarefa do MONITOR
```

* Macros não construidas.

6.2.2. O Módulo Monitor

A tarefa MONITOR recebe o controle da simulação e tem a função de fornecer um meio do dedicado interagir com o usuário, operador ou programador do sistema.

O monitor desenvolvido inteiramente no modo protegido utiliza a interface serial RS 232 para comunicar-se com um PC que emula um terminal conectado na outra extremidade. Ao receber o controle da CPU, a tarefa MONITOR envia um *prompt* para o terminal sinalizando que está esperando um comando para executar. Ao receber um comando válido, o dedicado o executa, e espera um novo comando. Os comandos para o **depurador** e **carregador** foram embutidos no MONITOR, permitindo que estas ferramentas estejam disponíveis para o ambiente simulado.

Durante a simulação do ambiente, o **carregador** foi utilizado, através do comando R do MONITOR, para receber o aplicativo exemplo EMBED1 desenvolvido pelo método proposto na seção 5.4.1. O cabeçalho necessário para este aplicativo foi construído e apresentado na figura 6.6. Nele são especificados, por exemplo, os endereços para o carregamento de cada segmento do aplicativo. Foram definidos os endereços 125000h para o segmento de dados e 135000 para o segmento de códigos. Os comandos do **depurador** foram exaustivamente utilizados nos dois aplicativos exemplos carregados no ambiente - APL1 e EMBED1.

```
EXTRA SEGMENT

nome      DB 'Embed1$'
num_seg   DB 2

          DW  Dados_Len
          DW  5000h
          DB  12h
          DB  92h
          DB  0
          DB  0

          DW Cod_Len
          DW  5000h
          DB  13h
          DB  9Ah
          DB  0
          DB  0

EXTRA ENDS
```

Figura 6.6 - Cabeçalho do aplicativo exemplo EMBED1.

6.2.3. Os Arquivos INCLUDE do Ambiente Dedicado

São arquivos que contêm códigos complementares para o ambiente dedicado. Foram criados 3 arquivos INCLUDE: ISR.INC, MACROS.INC e APOIO.INC.

O arquivo ISR.INC contém todas as rotinas de serviço de interrupção (ISRs) implementadas pelo software dedicado: o Mini_DOS, as interrupções e as exceções. O arquivo MACROS.INC contém todas as macros disponíveis como ferramentas de desenvolvimento, e o arquivo APOIO.INC é constituído somente por rotinas de apoio utilizadas pela tarefa MONITOR.

6.3. Configuração para a Placa do Dedicado

A configuração proposta para desenvolver software para execução na EPROM do dedicado é mostrada na figura 6.7. Há basicamente duas diferenças fundamentais entre

esta configuração e aquela da figura 5.4: a substituição do carregador do DOS pelo **realocador externo** e a presença do GRAVADOR de EPROM e do EMULADOR de EPROM.

A ferramenta **realocador externo** foi discutida ba seção 5.3.2. O GRAVADOR de EPROM e o EMULADOR de EPROM serão discutidos brevemente a seguir. A utilização das ferramentas no próprio ambiente dedicado, é realizada da mesma forma daquela utilizada na simulação com o PC, não necessitando de qualquer alteração para sua gravação em EPROM.

6.3.1. O Gravador de EPROM e o Emulador de EPROM

Após a etapa de realocação externa, o arquivo .BIN gerado pode seguir dois caminhos distintos: um GRAVADOR de EPROM ou um EMULADOR de EPROM. O primeiro é usado para gravar diretamente o programa numa EPROM para inserí-la de imediato na placa do dedicado. O segundo caminho passa por um conversor de formatos para enviá-lo ao EMULADOR de EPROM que é conectado por sua vez ao soquete da EPROM na placa. Esta alternativa pode ser explorada para testar e depurar os softwares de sistema e aplicativos antes da gravação da EPROM definitiva. A figura 6.8 mostra os dois caminhos para o arquivo ".BIN".

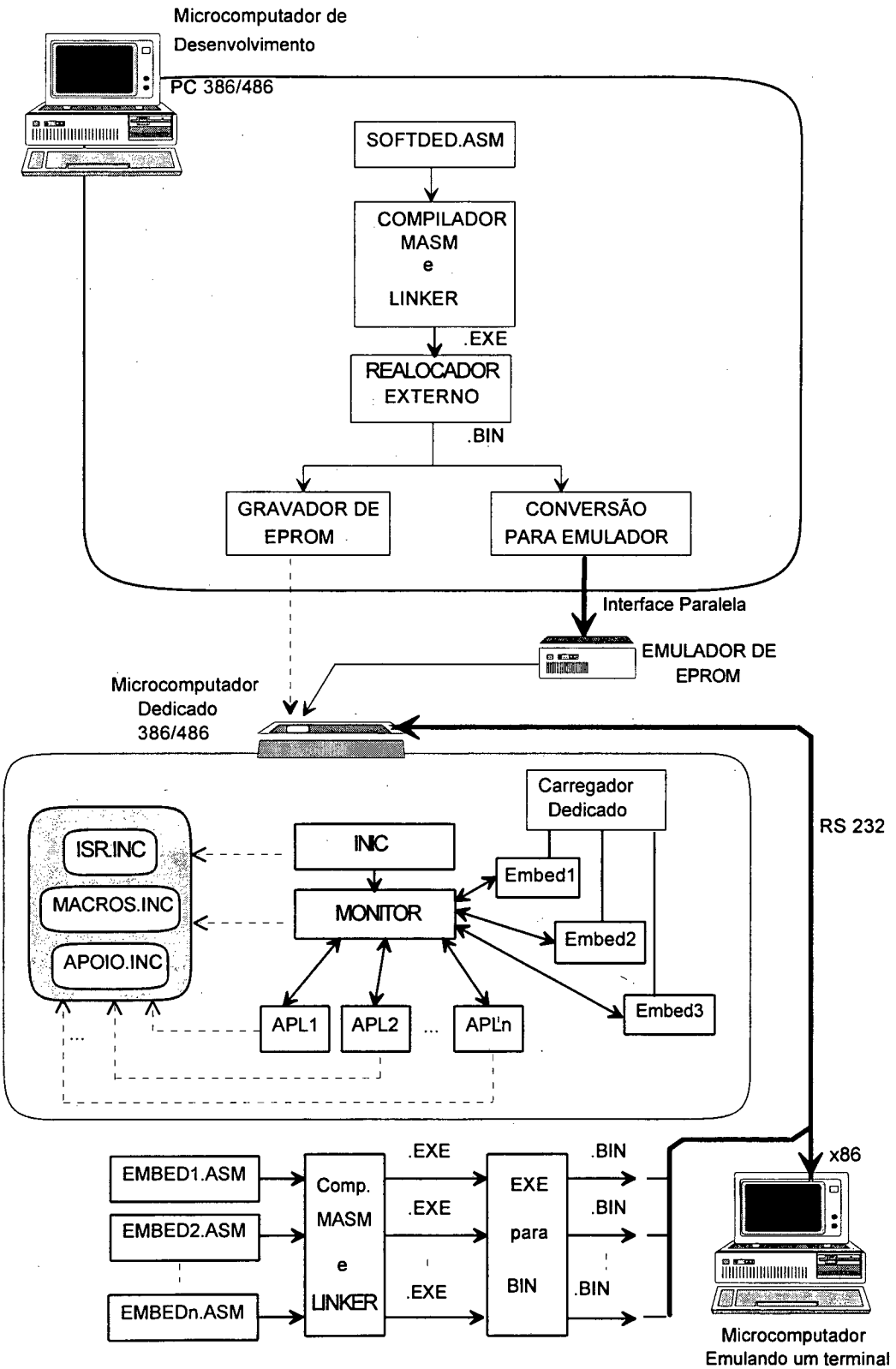


Figura 6.7 - Configuração para placa do dedicado.

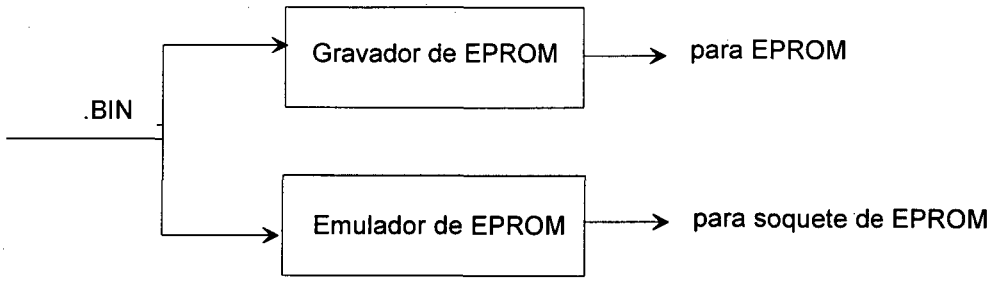


Figura 6.8 - Os dois caminhos para o arquivo ".BIN".

7. Conclusões e Recomendações

7.1. Conclusões

O trabalho apresentado descreveu a implementação de ferramentas para desenvolvimento de software para microcomputadores dedicados utilizando os processadores 80386/80486.

Cinco grupos de ferramentas foram implementadas: construtor, realocador, carregador, Mini-DOS e depurador.

As ferramentas foram utilizadas num microcomputador PC 386 compatível que simulava o ambiente dedicado, pois a CPU é a mesma nos dois casos. Dois aplicativos exemplos foram construídos, nos quais, as ferramentas desenvolvidas apresentaram desempenho bastante satisfatório. A simulação do carregamento e realocação de segmentos de EPROM para RAM no ambiente dedicado foi feita copiando alguns segmentos para uma área de memória acima de 2M bytes no PC. O acesso posterior a estes segmentos mostrou a funcionalidade das ferramentas. A utilização do Mini-DOS, depurador e carregador, no ambiente simulado, foi feita de forma semelhante daquela que seria utilizada se elas estivessem gravadas em EPROM no dedicado.

As ferramentas têm uma concepção simples, permitindo comprovar que o desenvolvimento de software no modo protegido do 80386 não possui alta complexidade. Assim, o desenvolvimento pode ser realizado sem a utilização de kits comerciais, caros, complicados e transparentes para o usuário.

A implementação das ferramentas foi realizada tendo em vista sua utilização pela linguagem Assembly. Entretanto a adaptação das mesmas para a linguagem C não constitui um problema, pois a metodologia de funcionamento é essencialmente a mesma.

Um manual de utilização das ferramentas está disponível numa versão preliminar, devendo ser concluído, em breve, sua versão final.

O uso do 80386 é justificado apenas para sistemas que possuem requisitos críticos de tempo ou de alto desempenho. Aplicações multitarefa podem tirar ainda mais proveito dos recursos disponíveis.

De uma forma geral, pode-se afirmar que os objetivos deste trabalho foram alcançados, utilizando o 80386 no modo protegido, como opção de desenvolvimento de software para microcomputadores dedicados. As ferramentas implementadas foram uma alternativa simples e rápida em relação à sistemática utilizada pelos kits de desenvolvimento pesquisados.

Por fim, destacamos que as ferramentas implementadas foram uma contribuição bastante significativa para a programação de dedicados, que tiram proveito da arquitetura sofisticada do 80386 operando no modo protegido.

7.2. Dificuldades

A principal dificuldade encontrada durante o desenvolvimento do trabalho restringiu-se à programação no modo protegido do 80386. Os manuais da Intel mostraram-se pouco elucidativos quanto à utilização dos recursos do modo protegido. As informações contidas nestes manuais não estão sistematizadas. A utilização correta dos recursos só foi possível após a análise exaustiva de códigos que acompanhavam alguns artigos sobre o assunto e através da implementação de pequenos trabalhos, nos quais, eram inseridos os recursos de forma gradual.

Uma dificuldade que certamente será encontrada para portar o software dedicado, juntamente com as ferramentas, numa EPROM de uma placa-mãe 80386, é a inicialização dos seus componentes. Os componentes presentes numa placa-mãe (*chipsets*) agregam um número de funções cada vez maior, necessitando de programação específica. Isto será alcançado através da leitura da EPROM que acompanha a placa, e da determinação do código referente à programação destes componentes.

7.3. Recomendações Futuras

Como continuação deste trabalho, sugere-se:

Aplicação das ferramentas numa placa PC 386 compatível através da sua gravação em EPROM juntamente com uma aplicação específica.

Implementação de um escalonador de tarefas, permitindo que cada aplicação carregada no dedicado seja constituída de uma ou mais tarefas (um programa autônomo com seu próprio contexto), que são executadas concorrentemente com as outras.

Um desmontador (*desassembler*) para a tarefa MONITOR, de tal forma que as próximas instruções a serem executadas pudessem ser enviadas ao terminal durante exceções de depuração.

Extensões do MONITOR: implementação de comandos para mostrar o conteúdo da GDT, IDT e TSS's.

Uma tarefa de rede de modo a possibilitar a comunicação entre o dedicado e outros sistemas de computação, bem como servir de meio para o carregamento de aplicativos.

Definição de uma LDT particular para cada aplicativo, permitindo determinar, automaticamente, os seletores utilizados pelo software (o primeiro seletor para o primeiro segmento, etc.).

O registro de exceções durante a execução do software dedicado é desejado. Isto possibilita ao projetista eliminar possíveis erros (*bugs*) que podem surgir durante o funcionamento normal do dedicado. O registro pode ser feito através da gravação em disco no terminal.

8. Referências Bibliográficas

[ALB90] Albrecht, Harald, "V86 - Virtual 8086 - Monitor for 80386 PCs", 1990.

[BAR92] Barron, Tovey, "Protected-mode Debugging Using In-circuit Emulators - Making a case for emulation", *Dr. Dobb's Journal*, February 1992.

[BRO94] Brown, Jonh F., "Embedded Systems Programming in C and Assembly", VNR, 1994.

[BUR94] Burgess, Richard, " MMURTL: Your Own 32-bit Operating System", *Dr. Dobb's Journal*, May 1994.

[COM88] Comer, Douglas, "Operating Systems Design Vol 1: The XINU Approach (PC Edition)", Prentice-Hall, 1988.

[CON94] Concurrent Sciences Inc, Informações de folders, 1994.

[COO91] Cook, Rick. "Embedded Systems in Control", *Byte,* June 1991.

[CRA87] Crawford, John H. and Gelsinger, Patrick P., "Programming the 80386", Sybex, 1987.

[DPM90] DOS Protected Mode Interface (DPMI), Protected Mode API For DOS Extended Applications, Version 0.9, 1990.

[DUN90] Duncan, Ray, "MSDOS Avançado - Guia do Usuário", Makron Books, 1990.

[ELE94] *Elektronik,* Vol 13, June 28 1994, pp 20-22.

[FER90] Ferri, Enrique H., "Introdução 80386/486 - Arquitetura e Componentes de Hardware", Erica, 1990.

[FRI90a] Fried, Stephen, "Accessing Hardware from 80386/486 Protected Mode Part I, Understanding the 386 architecture may simply be a matter of building on what you already know", *Dr. Dobb's Journal*, May 1990.

[FRI90b] Fried, Stephen, "Accessing Hardware from 80386/486 Protected Mode Part II, A 4-gigabyte memory model and features that control 80386/486 paging make FAR pointers obsolete", *Dr. Dobb's Journal*, June 1990.

[GRE89] Green, Tom, "80386/486 Protected Mode Multitasking - Putting the 386 to work under MS-DOS", *Dr. Dobb's Journal*, September 1989.

[INT87] Intel, "80386/486 System Software Writer's Guide", Intel Corporation, 1987.

[INT90a] Intel, "i80486 Microprocessor Programer's Reference Manual", Intel Corporation, 1990.

[INT90b] Intel, "i486 Microprocessor Hardware Reference Manual", Intel Corporation, 1990.

[INT91] Intel, "intel486 SX Microprocessor / intel 487 SX Math Coprocessor", Data Book, Intel Corporation, 1991.

[INT94] Intel, "386EX Embedded Microprocessor Hardware Reference", Intel Corporation, 1994.

[KNO92] Knoblaugh, Rick, "Your Own Protected-mode Debugger - A resident debugger for 80386/486/486 platforms", *Dr. Dobb's Journal*, September 1992.

[KOO89] Koopman, Philip J., "Stack Computers - The New Wave", Ellis Horwood, 1989.

[LEI93] Leinecker, Richard C., "Processor Detection Schemes - You don't have to write for the least-common denominator", *Dr. Dobb's Journal*, June 1993.

[MAR89] Margulis, Neal, "Advanced 80386/486 Memory Management - Paging is the 80386/486's answer to the memory management challenges for today's multitasking operating systems", *Dr. Dobb's Journal*, April 1989.

[MIL90] Miller, Alan, "Assembly IBM PC Técnicas de Programação", EBRAS Editora Brasileira, 1990.

[MIP90] MIPEPS, "Manual do Simulador de EPROM", MIPEPS plus, 1990.

[SCA88] Scanlon, Leo J., "8086/8088/80286 assembly language", A Brady Book, 1988.

[SCH90] Schulman, Andrew et al., "Undocumented DOS - A Programer's Guide to Reserved MS-DOS Functions and Data Structures", Addison-Wesley, 1990.

[SEG92] Segal, Bernardo, Nakajune, Cesar K. and Celestino, Sílvio A., "Conhecendo a Família 80486 - Hardware e Software", Erica, 1992.

[SIN90] Singh, Avtar and Triebel, Walter, " The 8086 and 80286 Microprocessors, Hardware, Software and Interfacing ", Prentice-Hall , 1990.

[SSI93] SSI - Systems & Software, Inc. "How to Get Embed with the 386", 1993.

[SSI94] SSI - Systems & Software, Inc. "Writing Romable Code for the 386, 1994.

[TZO93] Tzou, Shin-Yuan et al., "A Distributed Development Environment for Embedded Software", *Software Practice and Experience*, Vol 23, pp 1235-1248, November 1993.

[WIL90a] Williams, Al, "Roll Your Own DOS Extender: Part I - Develop your own 386 protected-mode applications", *Dr. Dobb's Journal*, October 1990.

[WIL90b] Williams, Al, "Roll Your Own DOS Extender: Part II - Under the hood", *Dr. Dobb's Journal*, November 1990.

[WIL90c] Williams, Al, "DOS + 386 = 4 Gigabytes!- Directly address 4 gigabytes of memory in DOS from your C or assembly languages applications", *Dr. Dobb's Journal*, July 1990.

Apêndice A - Macros

A.1. Construção de TSS

A macro Gera_Tss constrói um TSS preenchendo uma área já reservada para estas estruturas. Este acesso é realizado com a utilização de um *alias* para esta área passada para a macro como o número da tarefa a ser criada. Desta forma pode-se ter já prontos os descritores desses TSS's na GDT quando os mesmos forem sendo criados durante a execução do programa.

Declaração de uma estrutura para os TSS's:

```
TSSBLK STRUCT

Blink       DW   0,0    ; Back Link
S0PTR       DD   0      ; Ponteiro do Stack de nível 0
S0Sel       DW   0,0    ; Seletor do Stack de nível 0
S1PSel      DD   0,0    ; Ponteiro e Seletor do Stack de nível 1
S2PSel      DD   0,0    ; Ponteiro e Seletor do Stack de nível 2
CR3         DD   0      ; CR3 (para paginação)
EIP         DW   0,0    ; EIP
EFLAGS      DD   0      ; EFLAGS
EAX/EBX     DD   0,0    ; EAX e EBX
ECX/EDX     DD   0,0    ; ECX e EDX
SPTR        DD   0      ; ESP
EBP/ESI/EDI DD   0,0,0  ; EBP, ESI e EDI
ESSel       DW   0,0    ; Seletor de ES
CSSel       DW   0,0    ; Seletor de CS
SSSel       DW   0,0    ; Seletor de SS
DSSel       DW   0,0    ; Seletor de DS
```

```
FSSel       DW   0,0                    ; Seletor de FS
GSSel       DW   0,0                    ; Seletor de GS
LDTSel      DW   0,0                    ; Seletor da LDT
            DW    0                     ; Debug-Bit 0
EndBitMap   DW    $+2- OFFSET Blink     ; Offset I/O-Permission-Bitmap rel.
IOP         DB    8192 DUP (0)
            DB   0FFH

TSSBLK      ENDS
```

Declaração de variáveis do tipo TSSBLK

Foram declaradas 5 variáveis do tipo TSSBLK sem nenhuma inicialização. O preenchimento de cada TSS será realizado posteriormente no momento de sua criação pela macro *Gera_TSS*. Todos os TSS's estarão dentro de um único segmento. Isto facilita a inicialização pois será necessário apenas um segmento de dados *alias* que englobe todo o segmento dos TSS's.

```
TSSSEG      SEGMENT PARA PUBLIC 'DATA32' USE16

        TSS0  TSSBLK < >
        TSS0END   EQU $

        TSS1  TSSBLK < >
        TSS1END   EQU $

        TSS2  TSSBLK < >
        TSS2END   EQU $
```

```
        TSS3  TSSBLK < >
        TSS3END   EQU $

        TSS4  TSSBLK < >
        TSS4END   EQU $

TSSEG           ENDS
```

Macro para gerar os labels TSS0, TSS1,.....,TSS9

```
        TSSLabel    MACRO  num

                    TSS&num

                    ENDM
```

Macro para preenchimento de uma TSS:

Esta é a macro para a inicialização de um TSS:

```
Gera_Tss MACRO TSS_Num, CSEL, DSEL, SSEL, SPPTR, IPPTR, SP0SEL,
SP0LIM, EFLAGSS, LDTSEL, ENDBITMAP

MOV    AX,Sel_TSS_alias              ; alias para o segmento que contém todos os
Tss's
MOV    DS,AX
MOV   BX, OFFSET TSSLabel%(TSS_Num)
MOV    WORD PTR DS:[BX].TSSBLK.CSSel,CSEL       ; seletor do CS
MOV    WORD PTR DS:[BX].TSSBLK.DSSel,DSEL       ; seletor do DS
```

```
MOV    WORD PTR DS:[BX].TSSBLK.SSSel,SSEL       ; seletor do SS
MOV    WORD PTR DS:[BX].TSSBLK.SPTR,SPPTR       ; ponteiro do stack
MOV    WORD PTR DS:[BX].TSSBLK.EIP,IPPTR        ; ponteiro de instrução
MOV    WORD PTR DS:[BX].TSSBLK.S0Sel,SP0SEL     ; seletor do stack de nível 0
MOV    WORD PTR DS:[BX].TSSBLK.S0PTR,SP0LIM     ; ponteiro do stack de nível 0
MOV    DWORD PTR DS:[BX].TSSBLK.EFLAGS1,EFAGSS  ; EFLAGS (32 bits)
MOV    WORD PTR DS:[BX].TSSBLK.LDTSel,LSEL      ; seletor da LDT
```

ENDM

onde TSS_Num é o número da tarefa a ser criada podendo ser de 0 a 9. O seletor de *alias* (TSS_alias) é único para todos os TSS's.

Como usar:

Gera_TSS *TSS_N, TASK_CODE_SEL, TASK_DATA_SEL, TASK_STACK_SEL , TASK_TOS, OFFSET TASK_START, TASK_STACK_SEL, TASK_TOS, TASK_FLASGS, TASK_LDT_SEL, TASK_BITMAP*

CONSIDERAÇÕES:

- Os descritores dos TSS's já devem ter sidos construídos na GDT antes mesmo da inicialização dos mesmos. Caso estes TSS's não-inicializados forem utilizados, ocorreriam consequências imprevisíveis.

- Há necessidade de reservar também memória para as próprias estruturas das TSS's. Isto causaria desperdício de memória caso elas não forem completamente utilizadas.

A.2. Construção de Descritor na GDT

Esta macro constrói na GDT um descritor de qualquer tipo. É especialmente útil para a construção ou ajuste de descritores após o carregamento de um segmento para uma área de memória qualquer, ou seja, no momento de seu carregamento. Assume-se que a GDT está numa área de dados perfeitamente acessada por um determinado seletor.

Gera_Desc **MACRO** BASE, OFFSET_INI, LIMITE, TAMANHO, DIREITOS, SELETOR

Onde,

BASE é o endereço base do segmento (32 bits)
OFFSET_INI é o deslocamento do início do segmento
LIMITE é o limite do segmento em bytes (32 bits)
TAMANHO é o tamanho do segmento (0=16 bits 1=32 bits) representado por um byte
DIREITOS são os direitos de acesso e tipo de segmento, representado por um byte podendo ser um dos casos abaixo:

DADOS_SL	EQU	90H	Dados somente para leitura
DADOS_LE	EQU	92H	Dados para leitura e escrita
PILHA_SL	EQU	94H	Pilha somente para leitura
PILHA_LE	EQU	96H	Pilha para leitura e escrita
CÓDIGO_EX	EQU	98H	Código somente para execução
CÓDIGO_LE	EQU	9AH	Código para leitura e execução
CÓDIGO_CE	EQU	9CH	Código conforming para execução
CÓDIGO_CL	EQU	9EH	Código conforming para leitura e execução
DESC_TSS	EQU	89H	TSS

SELETOR é o seletor do descritor. A princípio serão disponíveis uma quantidade pré-determinada de descritores:

```
        SEL1, SEL2, SEL3,..., SELN
        PUSHAD
        MOV     AX,DATA_SEL
        MOV     ES,AX
        MOV     EDX,OFFSET GDT      ; Endereço da GDT para ES:EDX
        MOV     SI,SELETOR
        MOVZX   ESI,SI               ; Move SELETOR para ESI e
        SHR     SI,3                 ; ajusta para índice na GDT
        MOV     AL,TAMANHO           ; Ajusta TAMANHO para posição correta
        SHL     AL,6                 ; no descritor
        MOV     ECX,LIMITE
        JBE     LIMITE_OK            ; G caso maior
        SHR     ECX,12               ; Divide por 4096 e seta bit G
        OR      AL,80H
LIMITE_OK:
        MOV     ES:[EDX+ESI*8],CX    ; Preenche o LIMITE no descritor
        SHR     ECX,16
        OR      CL,AL
        MOV     ES:[EDX+ESI*8+6],CL
        MOV     BX,BASE
        MOVZX   EBX,BX
        SHL     EBX,4                ; Preenche a BASE no descritor
        ADD     EBX,OFFSET_INI
        MOV     ES:[EDX+ESI*8+2],BX
        SHR     EBX,16
        MOV     ES:[EDX+ESI*8+4],BL
```

```
MOV     ES:[EDX+ESI*8+7],BH
MOV     AH,DIREITOS          ; Preenche o byte de DIREITOS de acesso

MOV     ES:[EDX+ESI*8+5],AH ; no descritor
POPAD
```

ENDM

Como usar:

Gera_Desc *SEG TASK_CODE,0, TASK_CODE_LEN, 0, CÓDIGO_LE, TASK_CODE_SEL*

CONSIDERAÇÕES:

- A macro Gera_Desc monta descritores somente na GDT, mas para ser utilizada também para uma LDT pode-se adaptá-la para receber o endereço da tabela desejada em ES:EDX.

A.3. Construção de Gates

Esta macro tem por objetivo construir *gates* na IDT para as rotinas de serviço de interrupção e exceção do dedicado. Pode ser utilizada para criar *gates* de interrupção, *gates* de tarefa e *gates* de *trap*.

Gera_gate **MACRO** N,SELETOR_SEG_ISR, OFFSET_DE_ENTRADA, DIREITOS

Onde,

N é o número da interrupção ou exceção a ser gerada na IDT.

SELETOR_SEG_ISR é o seletor do segmento da rotina de serviço da interrupção ou exceção.

OFFSET_DE_ENTRADA é o ponto de entrada da rotina de serviço.

DIREITOS são os direitos de acesso (DPL,P) e o tipo de *gate* a ser construído.

```
        PUSHAD
        MOV     AX, DATA_SEL
        MOV     ES,AX
        MOV     AL,BYTE PTR ES:[IDT_PTR+4]
        MOVXZ   EAX,AL
        SHL     EAX,16
        MOV     AX,WORD PTR ES:[IDT_PTR+2] ;Lê o endereço da IDT em
        MOV     EDX,EAX                    ;IDT_PTR
        MOV     AX,UNIVERSE_SEL
        MOV     ES,AX                      ;Segmento de 4GB para ES
        MOV     SI,N
        MOVZX   ESI,SI
        MOV     EBX,OFFSET_DE_ENTRADA
        MOV     ES:[EDX+ESI*8],BX
        MOV     CX,SELETOR_SEG_ISR
        MOV     ES:[EDX+ESI*8+2],CX
        MOV     AX,DIREITOS
        MOV     ES:[EDX+ESI*8+4],AX
        SHR     EBX,16
        MOV     ES:[EDX+ESI*8+6],BX
        POPAD

ENDM
```

Como usar:

Gera_gate *09H,ISR_TSS_SEL,0,0E500H*

CONSIDERAÇÕES:

- Esta macro deve ser utilizada após a construção do descritor do segmento de código da rotina de serviço de interrupção.
- No caso de usar a macro para criar *gate* de interrupção ou de *trap* deve-se passar o seletor do TSS e o offset de entrada. No caso de *gate* de tarefa apenas o seletor do TSS interessa desprezando-se o offset (nulo).

A.4. Carregamento de um Trecho de Programa/Dados para uma Área de Memória Qualquer

Esta macro copia um trecho de programa ou dados para uma região de memória desejada qualquer. O objetivo principal desta macro é copiar segmentos da memória EPROM para RAM. Considera-se que o descritor associado já esteja corretamente definido na GDT.

Copia_Seg_RAM **MACRO** SEG_BASE_ROM, OFFSET_INI, LIMITE, POS_RAM

Onde,

SEG_BASE é o endereço base do segmento que contém o trecho a ser transferido para a área de memória desejada.
OFFSET_INI é o offset do trecho a ser transferido em relação à base.
LIMITE é o tamanho do trecho (número de bytes) a ser transferido para a área de memória desejada.

POS_RAM é endereço destino em RAM.

```
MOV       CX,Limite
MOVZX     ECX,CX
MOV       SI,Seg_base_ROM
MOVZX     ESI,SI
SHL       ESI,4
MOV       AX,OFFSET_INI
MOVZX     EAX,AX
ADD       ESI,EAX
MOV       EDI, POS_RAM
MOV       AX,SEL_UNIVERSAL
MOV       DS,AX
MOV       ES,AX
REP MOVS  BYTE PTR [ESI], BYTE PTR [EDI]
```

ENDM

Como usar:

Copia_Seg_RAM *SEG DATA,0,5000h,200000*

Copia 5000h bytes do segmento de dados a partir do início para o endereço 2M.

A.5. Carregamento de uma Tabela de Sistema para uma Área de Memória Qualquer

Esta macro copia uma tabela de sistema para uma região de memória desejada qualquer e altera o registrador da tabela com o novo endereço. O objetivo principal desta macro é copiar as tabelas da memória EPROM para RAM.

Copia_Tab_RAM **MACRO** TABELA, POS_RAM

Onde,

TABELA é a tabela a ser copiada.

POS_RAM é o endereço destino da tabela em RAM.

```
CLI                                  ; Interrupções desabilitadas
MOV     SI,dados
MOVZX   ESI,SI
SHL     ESI,4
MOV     EBX,ESI
MOV     AX,OFFSET TABELA
MOVZX   EAX,AX
ADD     ESI,EAX                      ; Calcula endereço origem em ESI
MOV     ECX,&TABELA&_LEN             ; Tamanho da tabela
MOV     EDI, POS_RAM                 ; Endereço destino em EDI
MOV     EAX,OFFSET &TABELA&_PTR
ADD     EBX,EAX
MOV     AX,SEL_UNIVERSAL             ; DS e ES --> 4G Bytes
MOV     DS,AX
MOV     ES,AX
MOV     DWORD PTR DS:[EBX+2],POS_RAM ; Atualiza valor do
                                     ;registrador
L&TABELA& FWORD PTR DS:[EBX]         ; Atualiza o registrador da
REP MOVS  BYTE PTR [ESI], BYTE PTR [EDI]   ; tabela e copia....
STI
```

ENDM

Como usar:

Copia_Tab_RAM *GDT,300000*

Copia a GDT do segmento de dados para o endereço 3M.

CONSIDERAÇÕES:

- É necessário que as tabelas do sistema estejam num mesmo segmento de dados para a macro funcionar corretamente, caso sontrário, será necessário passar para a mesma o segmento da tabela.

Apêndice B - Os Processadores 80386/80486

Os Processadores 80386/80486

Neste apêndice são descritos os recursos existentes no modo protegido da arquitetura dos processadores 80386/80486 quanto à gerência de memória, proteção, memória cache e capacidade de multitarefa.

B.1. Gerenciamento de Memória

A arquitetura do 80386/80486 no modo protegido de operação aumenta a capacidade de endereçamento linear de 1M bytes - limite imposto pelos 20 bits de endereço usados pelo modo real de operação - para 4G bytes (2^{32} bytes) , com a utilização de todos os 32 bits de endereço do processador. Além disso, é possível executar todo o software existente do 8086 e 80286 enquanto se utiliza um sofisticado gerenciamento de memória e um mecanismo de proteção [INT90b].

A principal diferença entre os modos real e protegido de operação no tocante ao gerenciamento de memória é o mecanismo de endereçamento.

B.1.1. Segmentação

O modo protegido de operação utiliza duas componentes para formar o **endereço lógico**: um **seletor** de 16 bits- usado para determinar o endereço base do segmento - e um **endereço efetivo** de 32 bits que representa o offset. O endereço lógico é transformado em **endereço linear** de 32 bits somando-se o endereço base do segmento com o endereço efetivo. O endereço linear é usado como endereço físico, ou se a paginação estiver habilitada, o mecanismo de endereçamento mapeia o endereço linear de 32 bits em um endereço físico de 32 bits.

Para calcular o endereço base de um segmento o mecanismo de endereçamento procede da seguinte maneira: o seletor é usado para especificar um índice em uma das tabelas de descritores do sistema, a GDT ou LDT, que contém os 32 bits de endereço da

base do segmento. O endereço linear é formado somando-se esse endereço obtido da tabela com o offset como mostra a figura B.1.

B.1.2. Paginação

A paginação fornece um mecanismo de endereçamento adicional operante apenas no modo protegido do 386/486 e no modo virtual 86 (VM86). É usada para simular um espaço maior de endereçamento não segmentado usando pequenos espaços de endereços fragmentados e o recurso de armazenamento em disco.

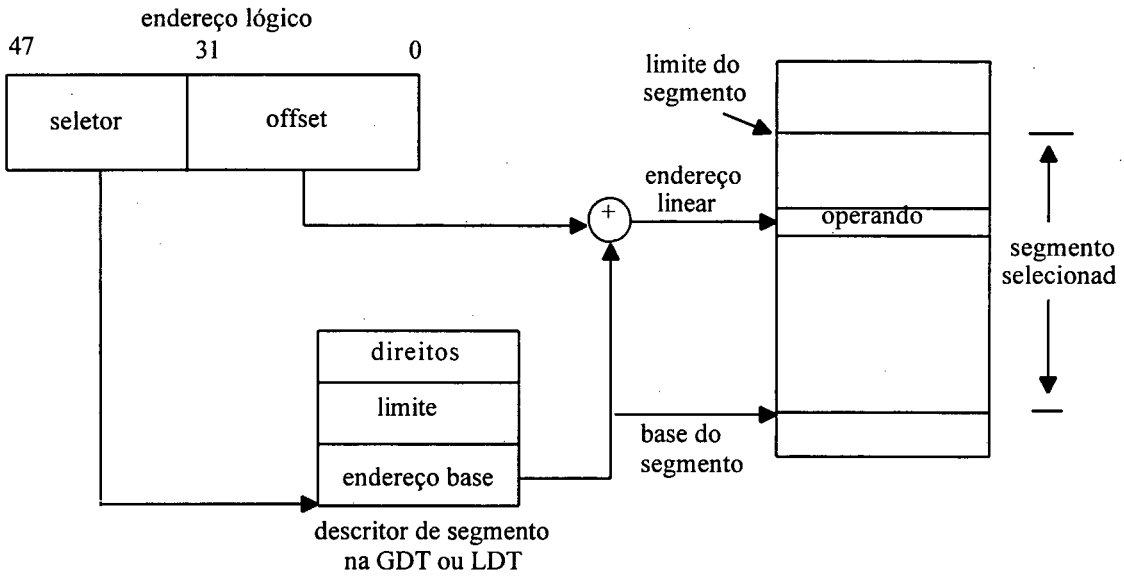


Figura B.1 - Mecanismo de endereçamento utilizando segmentação [INT87].

A paginação possibilita o acesso a estruturas de dados maiores que o espaço de memória disponível mantendo parte armazenada em memória e parte armazenada em disco. Com a paginação habilitada a memória é dividida em trechos chamados de páginas com tamanhos iguais a 4K bytes [INT87]. Usando esse recurso pode-se armazenar em disco as páginas menos solicitadas e carregá-las para a memória quando necessitadas através do mecanismo de "swap" de páginas - a página armazenada em disco é trocada por uma página da memória.

O mecanismo de paginação traduz o endereço linear proveniente da unidade de segmentação em um endereço físico (figura B.2).

Uma página de memória é endereçada com o auxílio de 2 tabelas: tabela de diretório de páginas e tabela de páginas. A tabela de diretório de páginas (1°. nível) endereça até 1K tabelas de páginas (2°. nível) e estas até 1K páginas na memória física. Todas as tabelas de páginas endereçadas pela tabela de diretório de páginas pode endereçar 1M páginas. Portanto, se cada página contém 4K bytes, todo o espaço de endereçamento de 4G bytes pode ser acessado utilizando-se as 2 tabelas (1K x 1K x 4K).

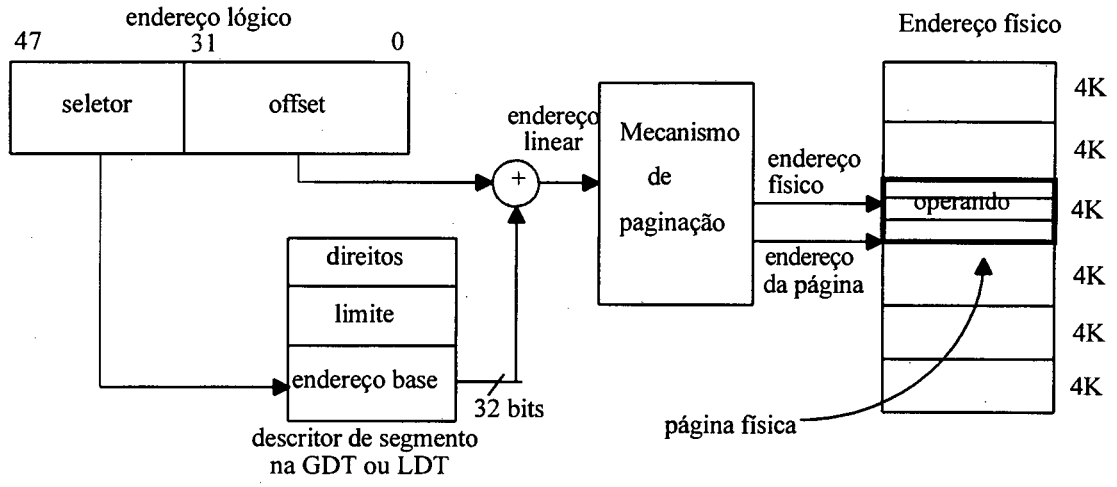


Figura B.2 - Mecanismo de endereçamento utilizando segmentação e paginação [INT90a].

O endereço físico da tabela de diretório de páginas corrente é armazenado no registrador CR3, também chamado de registro base de diretório de páginas (PDBR). Os 10 bits mais significativos do endereço linear é usado para indexar uma entrada na tabela de diretório de páginas. Esta entrada define o endereço físico de uma tabela de páginas. Os 10 bits seguintes do endereço linear é usado então para indexar uma entrada na tabela de páginas, a qual contém o endereço físico da página endereçada. Os 12 bits menos significativos do endereço linear são finalmente utilizados para endereçar uma posição dentro da página de 4K bytes. A figura B.3 ilustra o mecanismo de tradução de páginas.

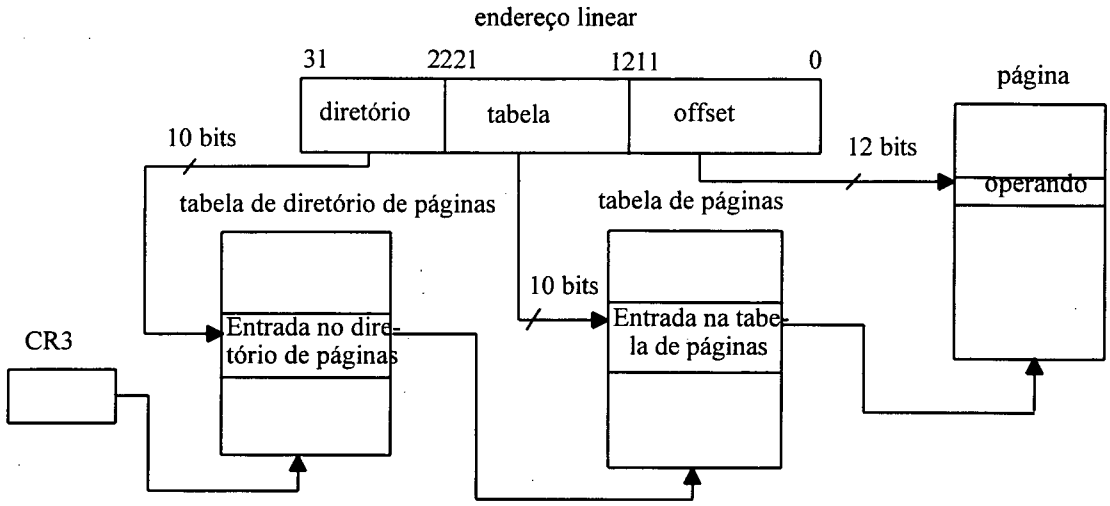


Figura B.3 - Mecanismo de tradução de páginas [INT90a].

Pode-se utilizar um mesmo diretório de páginas para todas as tarefas num sistema multitarefa ou pode-se ter um diretório de páginas específico para cada tarefa. Neste caso, cada tarefa deve inicializar o registrador CR3 com o endereço físico de sua tabela de diretório de páginas.

A paginação constitui-se em um recurso extremamente útil em situações onde é desejada a execução de um programa em uma área de memória física diferente daquela em que o programa se referencia. Através da alteração do endereço físico de execução na entrada da tabela de páginas referente ao endereço linear utilizado pelas páginas do programa, é possível executá-lo fisicamente numa área qualquer, sem contudo alterar qualquer referência a endereços de segmento e offsets para o código. Desse modo o programa "pensa" que está executando numa área mas na verdade o processador endereça uma área completamente diferente.

A seguir é dado um exemplo de procedimento que pode ter bastante utilidade na programação de dedicados:

➢ Transferência de código ou dados para uma posição qualquer de memória

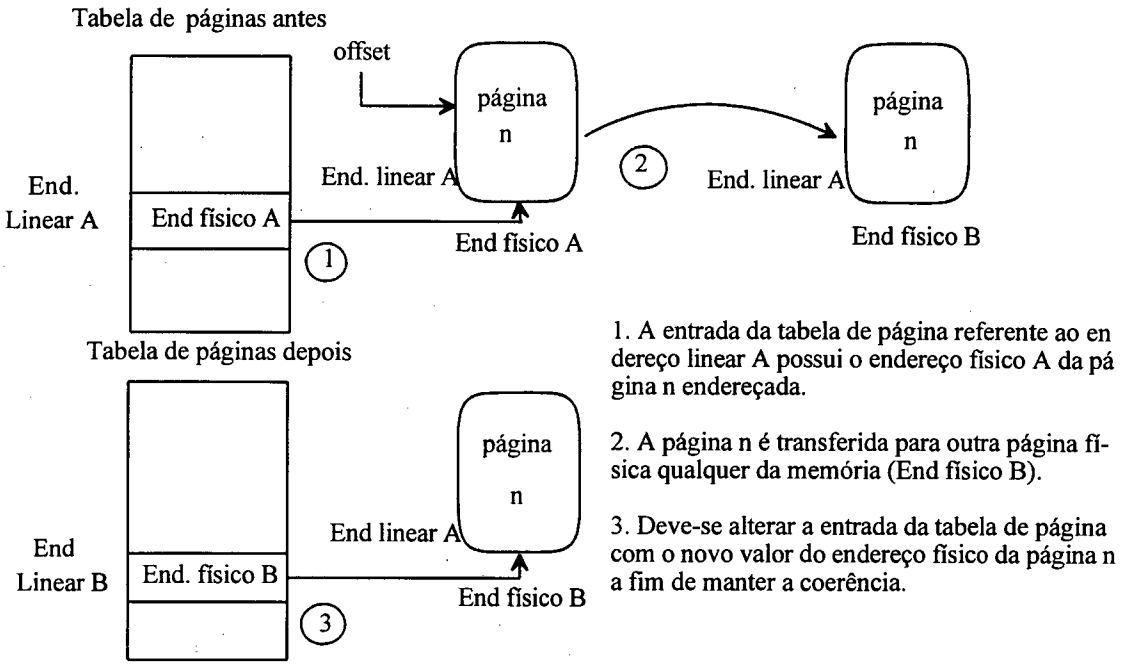


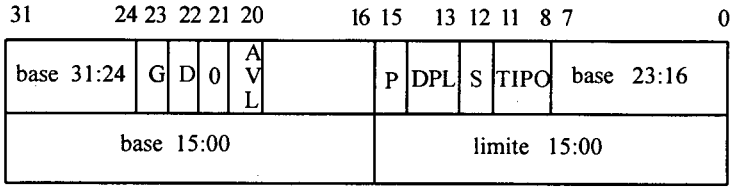
Figura B.4 - Exemplo de procedimento de alteração de endereço físico de páginas [INT91].

É interessante perceber que não foi feita qualquer alteração no descritor que define o endereço base do segmento que contém a página n, pois como sabemos, os descritores de segmento nas tabelas de descritores (GDT ou LDT) definem apenas o endereço linear de um segmento, o qual permanece inalterado neste caso.

Com este procedimento pode-se deslocar qualquer quantidade de páginas para outra região de memória sem se preocupar com os endereços que o programa contido nestas páginas referencia. O único cuidado que deve-se tomar é que a(s) página(s) deve(m) ser deslocada(s) sempre para um endereço alinhado de página (4K).

B.1.3. Descritores de Segmento

Descritores são estruturas de dados de 8 bytes que contêm atributos de uma área de endereçamento de memória, ou , mais simplesmente de um segmento [INT90a]. Esses atributos incluem 32 bits do endereço de base linear do segmento, 20 bits do limite e granularidade do segmento, nível de proteção, privilégios e tipo de segmento. A figura B.5 mostra o formato geral de um descritor de segmento:



BASE Endereço base do segmento
DPL Nível de privilégio do segmento
LIMITE Limite do segmento
S Tipo do descritor (0->sistema 1-> aplicação)
G Granularidade
P Segmento presente
TIPO Tipo do segmento
AVL Disponível para uso pelo software do sistema
D Tamanho default de operação(aplicado somente a descritores de segmento de código
0 -> segmento de 16 bits 1 -> segmento de 32 bits

Figura B.5 - Descritor de segmento [INT90a].

O campo *tipo* especifica se o segmento é de dados ou de códigos e se é somente para leitura, leitura e escrita (para segmentos de dados), somente para execução, para execução e leitura (no caso de segmento de código).

Granularidade - 0 - > limite é interpretado em unidades de 1 byte

- 1 - > limite é interpretado em unidades de 4K bytes (1 página).

B.1.4. Tabelas de Descritores

As tabelas de descritores definem todos os segmentos que são usados pelo 80386/80486. Há três tipos de tabelas que guardam descritores: tabela de descritores globais (GDT), tabela de descritores locais (LDT) e tabela de descritores de interrupções (IDT). Elas podem ter tamanho variando de 8 bytes (1 descritor) a 64 K bytes (8192 descritores) .

Cada tabela tem um registrador associado que contém seu endereço físico e limite.

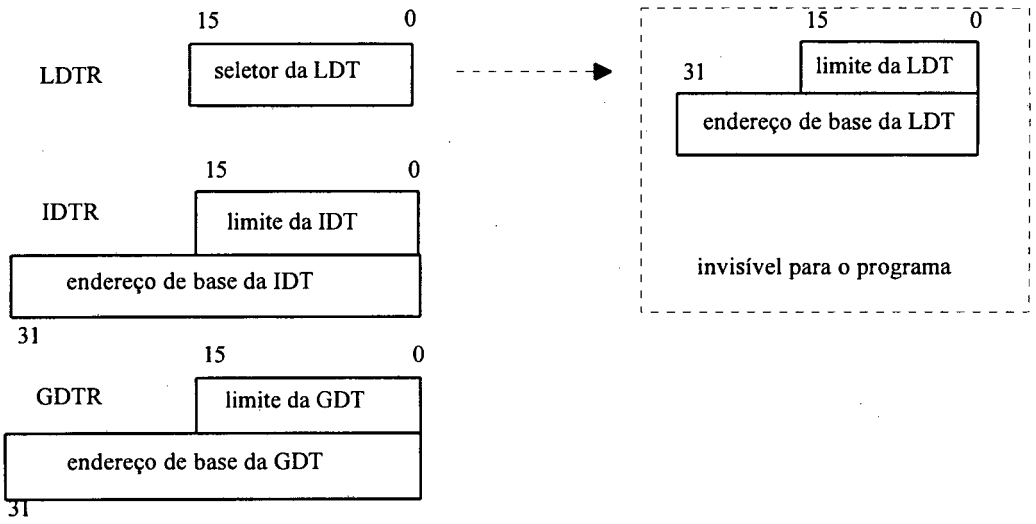


Figura B.6 - Registradores das tabelas de descritores.

A parte vísivel do LDTR ao contrário do GDTR e IDTR, contém apenas um seletor de 16 bits que indexa um descritor de LDT na GDT.

As instruções LGDT, LLDT e LIDT carregam o endereço de base e o limite da tabela correspondente para o registrador apropriado, e as instruções SGDT, SLDT e SIDT armazenam os valores do endereço de base e limite. Essas instruções só podem ser executadas pelo sistema operacional constituindo-se então em instruções privilegiadas - instruções que só podem ser executadas no nível de privilégio 0.

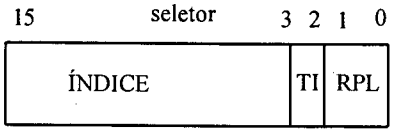
A tabela de descritores globais (GDT) pode ser utilizada por todas as tarefas do sistema e conter qualquer tipo de descritor de segmento exceto àqueles usados para tratamento de interrupções. Geralmente a GDT contém descritores de segmento de código e dados usados pelo sistema operacional, descritores de TSS (segmento de estado da tarefa) e descritores de LDT.

A tabela de descritores locais (LDT) fornece um mecanismo de isolar os segmentos de código e dados de uma tarefa de outra tarefa e do resto do sistema operacional, pois a cada tarefa pode-se associar uma LDT particular. Assim, um segmento não pode ser acessado por uma tarefa se o seu descritor não existe na GDT e na sua LDT [INT87].

A tabela de descritores de interrupções contém descritores que apontam para o local das 256 rotinas de tratamento de interrupções possíveis pelo 386/486. Cada interrupção utilizada pelo sistema deve ter uma entrada na IDT que pode ser do tipo gate de tarefa, gate de interrupção ou gate de trap. As entradas na IDT são referenciadas via instruções INT, vetores de interrupções externas ou exceções.

B.1.5. Seletores de Segmento

O seletor de segmento indexa um descritor de segmento em uma das 2 tabelas do sistema - GDT ou LDT. Um seletor de segmento identifica um descritor de segmento especificando uma tabela de descritores e o índice do descritor dentro da tabela. O processador multiplica o índice por 8 (o número de bytes de um descritor) e soma este valor com o endereço da base da tabela selecionada (usando o GDTR ou LDTR). Ver figura B.7.



ÍNDICE - Índice do descritor na tabela

TI - Indicador da tabela TI=0 -> GDT e TI=1 -> LDT

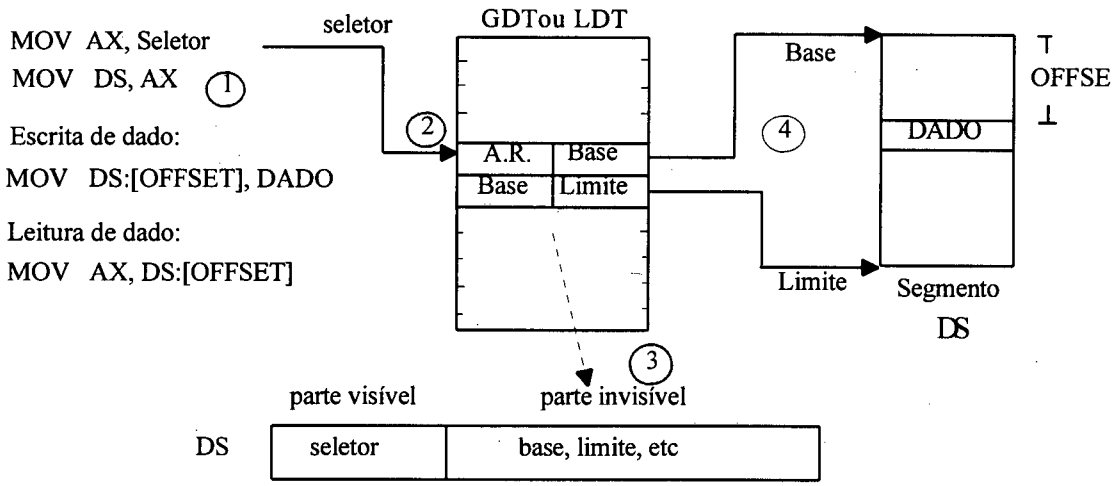
RPL - Nível de privilégio requisitado RPL=00 -> mais privilegiado RPL=11 -> menos privilegiado

Figura B.7 - Seletor de segmento.

Assim se o seletor tiver valor igual a 0028H trata-se do 6°. seletor da GDT (o primeiro têm índice igual a 0), ou seja,

0000 0000 0010 1000 -> Índice = 5 e TI = 0 (GDT) e RPL = 00

Exemplo de utilização de um seletor:



1. O registrador DS é carregado com um seletor.

2. O seletor indexa um descritor de segmento na GDT ou LDT (índice*8). ce

3. O registrador DS armazena os dados lidos do descritor de segmento.

4. O dado é acessado no segmento com o offset determinado pelo operando.

Figura B.8 - O uso de um seletor de segmento.

B.2. Proteção

O 80386/80486 possui 4 níveis de proteção que suportam as necessidades de um sistema operacional multitarefa de isolar e separar programas do usuário entre si e do sistema operacional. Os níveis de privilégio controlam o uso de instruções privilegiadas e o acesso a segmentos e a descritores de segmento.

Os 4 níveis de privilégio são mostrados na figura B.9 e são numerados de 0 a 3 sendo o nível 0 o mais privilegiado e o nível 3 o menos privilegiado.

B.2.1. Regras de Privilégio

O acesso a dados e código entre níveis de uma tarefa é controlado conforme as regras seguintes [INT91]:

- Os dados armazenados num segmento com nível de privilégio igual a **p** pode ser acessado somente por código executando pelo menos com o mesmo nível de privilégio **p.**
- Um segmento de código (procedimento) com nível de privilégio **p** só pode ser chamado por tarefas executando num nível de privilégio igual ou menor que o nível **p.**

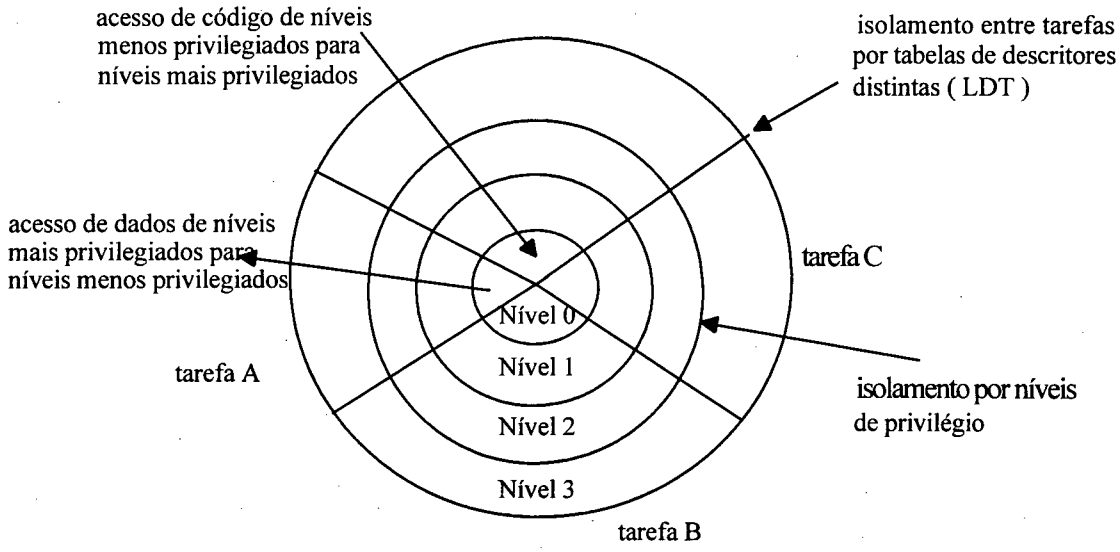


Figura B.9 - Os níveis de privilégio na hierarquia de proteção do 80386/80486 [INT91].

Em qualquer instante uma tarefa executa com um dos 4 níveis de privilégio. O nível de privilégio corrente (CPL) especifica o nível de privilégio da tarefa e só pode ser mudado por transferência de controle através de descritores de gates para segmento de código com nível de privilégio diferente. Assim, uma aplicação com CPL=3 pode chamar, por exemplo, uma rotina do sistema operacional que possui CPL=1 (via um *gate* de chamada) o que causaria a mudança de CPL da tarefa para 1 até que a rotina seja concluída, quando o CPL retornaria para 0. A figura B.10 mostra todos os acessos possíveis dentro do anel de proteção do 80386/80486.

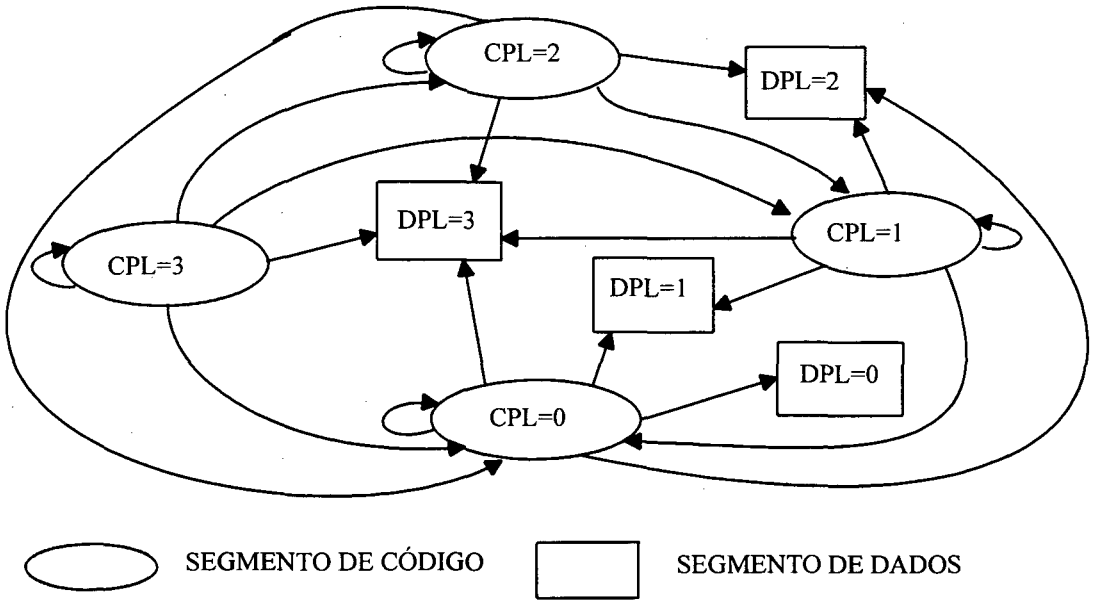


Figura B.10 - Os acessos permitidos na hierarquia de proteção do 80386/80486.

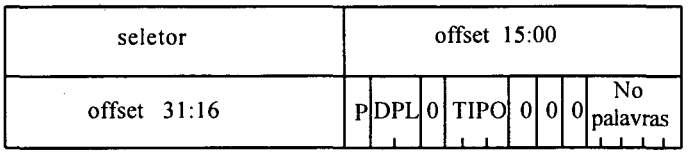
As transições entre segmentos de código são feitas por intermédio dos gates descritos a seguir e o acesso a segmento de dados são feitos através do carregamento de registradores de segmento de dados (DS, ES) com seletores de segmento. As transições entre segmentos de código que não alteram o valor de CPL podem ser feitas também através de carregamento do registrador de segmento de código (CS) com seletores de segmento.

Qualquer tentativa de acesso diferente dos ilustrados na figura causa uma exceção de proteção geral (exceção 13).

B.2.2. Descritores de Gates

Os descritores de *gates* são usados para controlar o acesso a segmentos de código. Eles podem ser do tipo *gate* de *trap*, *gate* de interrupção, *gate* de chamada e *gate* de tarefa. Os *gates* permitem aos projetistas de sistema controlar pontos de entrada para o sistema operacional. Os *gates* de chamada são usados para mudar o nível de privilégio de

uma tarefa (CPL), os *gates* de tarefa são usados para causar o chaveamento de tarefas, e os *gates* de interrupção e de *trap* são usados para definir rotinas de serviço de interrupção. O formato de um descritor de *gate* é mostrado na figura B.11.



TIPO 5 - task gate
C - call gate
E - interrupt gate
F - trap gate

DPL - menor nível de privilégio que uma tarefa deve ter para acessar o gate

No. palavras (0 a 31) - número de parâmetros para copiar do stack da rotina que está chamando para o stack da rotina chamada

SELETOR (16 bits) - seletor do segmento de código de destino ou seletor do segmento de estado da tarefa (gate de tarefa)

OFFSET (32 bits) - ponto de entrada dentro do segmento de código destino

Figura B.11 - Descritor de *gate*.

B.2.3. Gates de Chamada

Usados para transferir o controle do programa para um nível mais privilegiado. Consiste de 3 campos: o byte de acesso, um ponteiro longo - seletor+offset - que aponta para o início da rotina, e um contador de palavras que especifica quantos parâmetros devem ser copiados do stack do chamador para o stack da rotina chamada (destino). Esse campo é utilizado somente nos *gates* de chamada que causam mudança no nível de privilégio, nos outros *gates* este campo é ignorado.

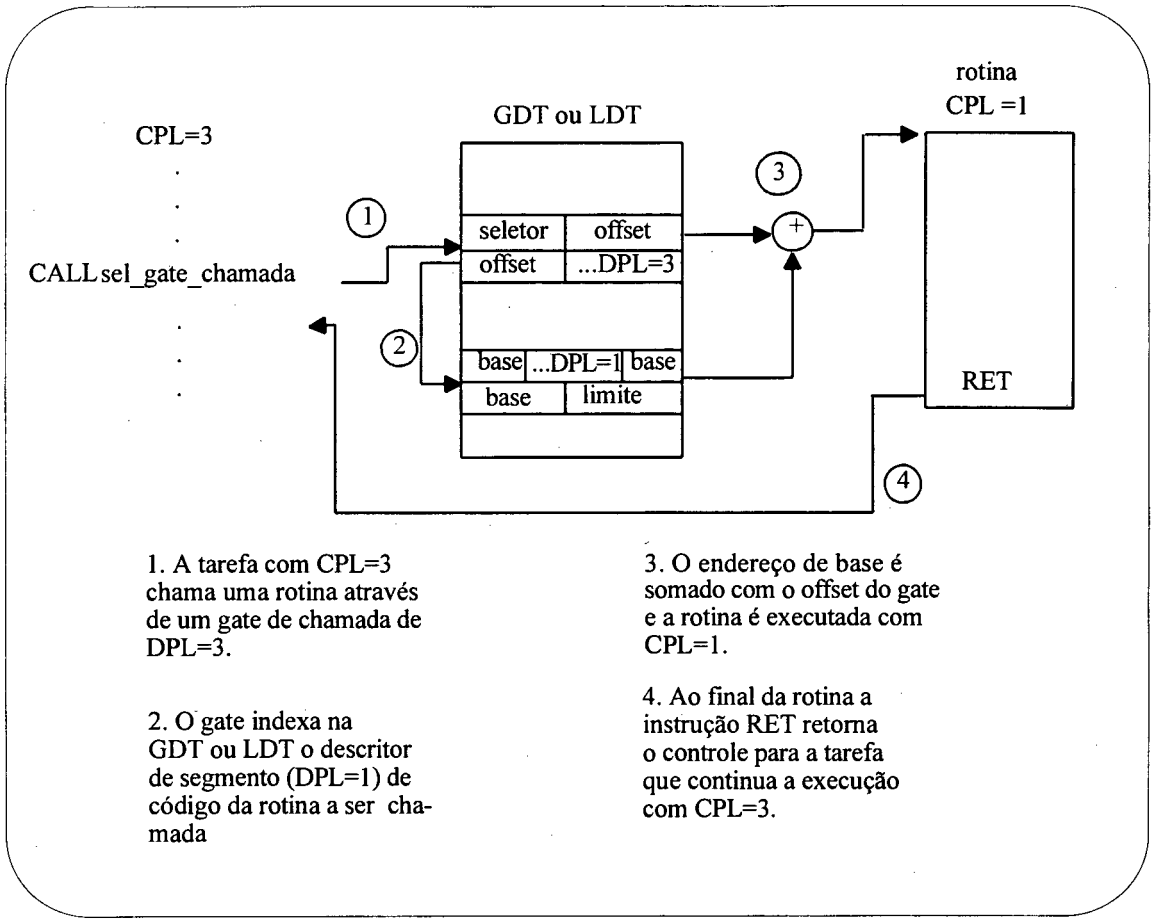


Figura B.12 - Utilização de um *gate* de chamada.

B.2.4. Gates de Interrupção e de Trap

Usam o seletor e offset do descritor de *gate* para definir o início da rotina de serviço de interrupção. A diferença é que o *gate* de interrupção desabilita as interrupções (reseta o bit IF do EFLAGS) e o *gate* de *trap* não.

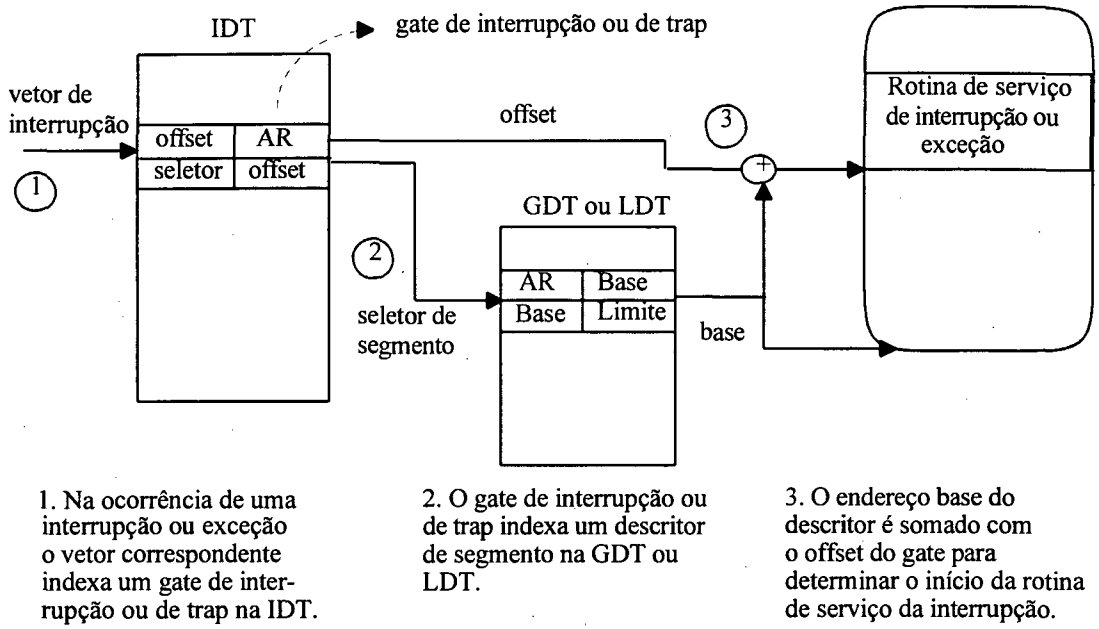


Figura B.13 - Utilização de um *gate* de interrupção ou de *trap*.

B.2.5. Gates de Tarefa

Usados para chavear de uma tarefa menos privilegiada para uma mais privilegiada. Os *gates* de tarefa necessitam apenas referenciar uma tarefa através de um seletor de segmento de estado de tarefa (TSS), desprezando o campo de offset do destino. Esses *gates* serão abordados com mais detalhes na seção de multitarefa.

B.2.6. Proteção à Nível de Páginas

Quando a paginação é habilitada um mecanismo adicional de proteção é aplicado às páginas de memória de 4K bytes. O mecanismo de paginação distingui entre 2 níveis de proteção: *usuário*, que corresponde ao nível 3 de proteção de segmentação, e *supervisor* que engloba os níveis de proteção restantes (2, 1 e 0).

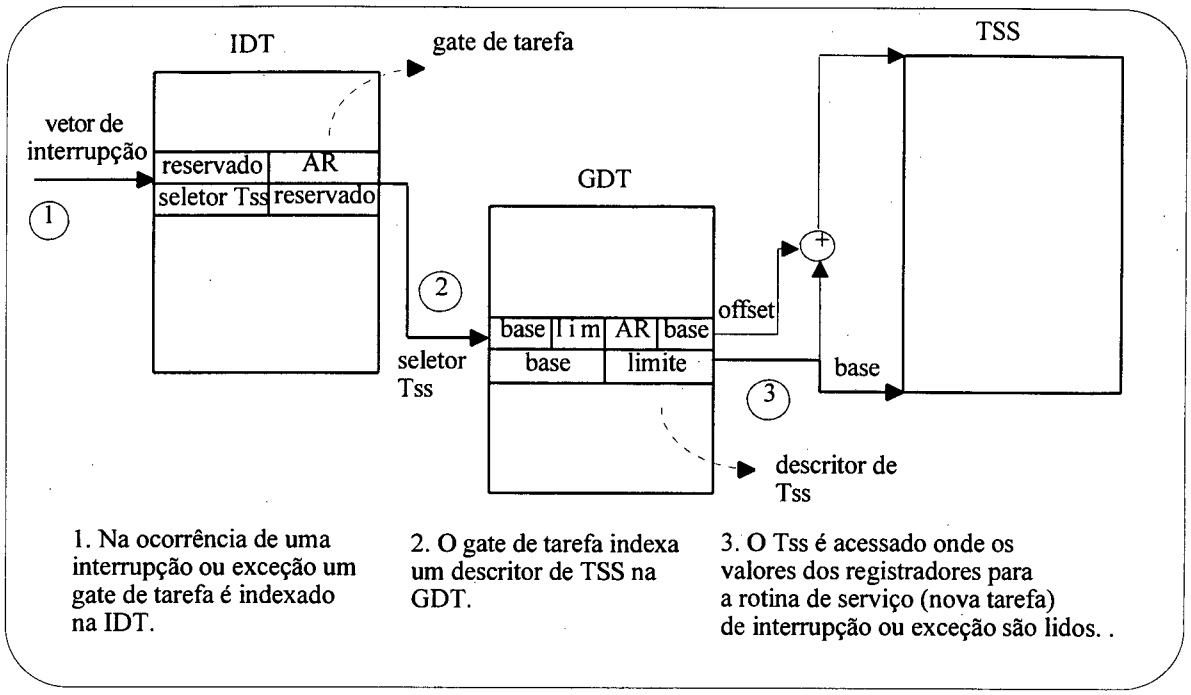


Figura B.14 - Utilização de um *gate* de tarefa.

Os bits R/W e U/S de uma entrada na tabela de diretório de páginas e de uma entrada na tabela de páginas (figura B.15) mais o bit WP no registrador EFLAGS, são usados conjuntamente para controlar o acesso às páginas de memória.

Os bits U/S e R/W nas entradas da tabela de diretório de páginas (primeiro nível) se aplicam a todas as páginas da tabela de páginas apontada por este diretório. Os bits U/S e R/W nas entradas das tabelas de páginas (segundo nível) se aplicam somente à página apontada por esta entrada da tabela. No caso de combinações diferentes nas 2 tabelas prevalece a combinação mais restritiva dos bits U/S e R/W para acessar uma determinada página [INT87].

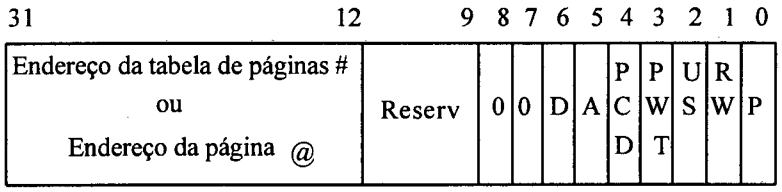
O bit WP do EFLAGS foi incorporado ao 80486 a fim de proteger as páginas definidas como *somente para leitura* contra acessos de escritas pelo *supervisor*. Quando WP=0, mantém-se compatibilidade com o 80386. Entretanto quando WP=1, o acesso a uma página *somente para leitura* (R/W=0) pelo supervisor causa exceção 14 (*page fault*).

A tabela B.1 mostra os efeitos dos bits WP, U/S e R/W para o acesso de uma página de memória.

U/S	R/W	WP	Acesso Usuário	Acesso Supervisor
0	0	0	Não	L/E/Exec
0	1	0	Não	L/E/Exec
1	0	0	L/Exec	L/E/Exec
1	1	0	L/E/Exec	L/E/Exec
0	0	1	Não	L/Exec
0	1	1	Não	L/E/Exec
1	0	1	L/Exec	L/Exec
1	1	1	L/E/Exec	L/E/Exec

L - leitura E - escrita Exec - execução

Tabela B.1 - Atributos de proteção a nível de páginas [INT87].



- para entradas na tabela de diretório de páginas
@ - para entradas na tabela de páginas

P - Presente

R/W - Escrita/Leitura

U/S - Usuário/Supervisor

PWT - Escrita na página desabilitada

PCD - "Cacheabilidade" da página desabilitada

A - Acessada

D - Suja

Reserv - Disponível para uso do programador do sistema

Figura B.15 - Entrada em uma tabela de diretório de páginas ou tabela de páginas.

B.3. Subsistema de Cache

Um dos recursos incorporados ao 80486 é o subsistema de cache interno. A memória cache é uma pequena porção de memória de alta-velocidade que se encaixa entre a CPU e a memória principal. Ela mantém cópias da memória principal que está em uso corrente com o objetivo de diminuir o tempo de acesso do microprocessador a instruções e dados requisitados, pois um ciclo de acesso à memória interna é executado muito mais rapidamente que um ciclo à memória principal, reduzindo assim, a latência de memória. Outra vantagem da utilização deste tipo de memória é que o número de acessos à memória dinâmica principal é consideravelmente reduzido facilitando a operação de ambientes multiprocessados com memória compartilhada [INT87]. O processador 80486 possui 8KBytes de memória cache interna para instruções e dados.

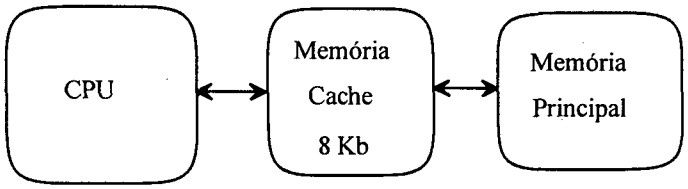


Figura B.16 - A memória cache.

A memória cache do 80486 consiste de um diretório (tag), e uma memória de dados. A cada ciclo de leitura ou escrita, o endereço de memória gerado é comparado com os tags e é verificado se os dados estão presentes na cache. Se estão na cache, houve um acerto na cache ou "cache hit" e o acesso é feito à unidade de memória associada com o tag. Se não estão na cache, houve uma perda na cache ou "cache miss". Neste caso, um ciclo de barramento é necessário para acessar a memória principal.

B.3.1. Estrutura da Cache

No 80486, a cache não é diretamente mapeada (apenas uma posição possível dentro da cache para o endereço e seu respectivo dado ou instrução). A organização da cache do 80486 é quadruplamente associativa (four-way set association). Há 4 posições

possíveis dentro da cache para armazenar um dado. Como mostrado na figura B.17 os 8KBytes são divididos em 4 grupos (ways) de dados, cada um contendo 128 conjuntos (sets) de 16 bytes, ou linhas de cache. Cada linha armazena o conteúdo de 16 endereços sucessivos de bytes da memória, iniciando com um endereço alinhado de 16.

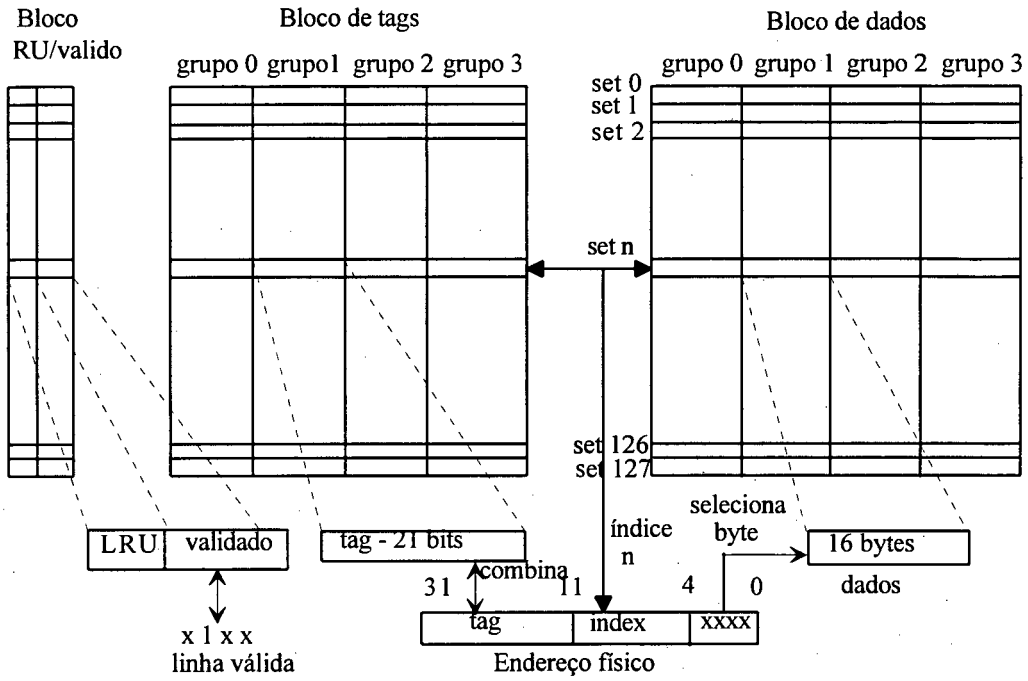


Figura B.17 - A estrutura da cache [INT87].

O mecanismo de endereçamento da cache é realizado dividindo-se os 28 bits mais significativos do endereço físico gerado pelo microprocessador em 3 partes como mostrado na figura B.17. Os 7 bits do campo *index* especifica o número do conjunto, um em 128, dentro da cache. Os 21 bits mais significativos compõem o campo de *tag*. Estes bits são comparados com os *tags* para cada linha no conjunto indexado, e indicam se a linha de cache de 16 bytes para aquele endereço físico está armazenada. Os 4 bits do campo *validado*, um para cada grupo pertencente a um conjunto (set) indica se os dados armazenados na cache naquele endereço físico são correntemete válidos.

B.3.2. Atualização da Cache

Na ocorrência de um "cache miss" na leitura, os 16 bytes que contém a informação requisitada são escritos na cache. O preenchimento das linhas da cache é feito apenas nas leituras perdidas (read misses) e nunca nas escritas perdidas (write misses). Cada escrita na cache não apenas atualiza a cache mas é também propagada através do barramento para a memória principal. Este tipo de procedimento é chamado **write-through**. As únicas condições onde os dados na cache são diferentes da memória é quando o ciclo de escrita na memória ainda não foi consumado devido a atrasos no barramento, ou quando um outro processador altera a memória principal mapeada na cache [INT91].

B.3.3. "Cacheabilidade" de Páginas

Dois bits de controle da cache, PCD e PWT, são defenidos em cada entrada do diretório de páginas e em cada entrada das tabelas de páginas (figura B.15). Estes bits são utilizados para gerenciar o processo de cache página a página. O bit PCD (page-level cache disable) controla a operação da cache interna. Tanto o bit PCD quanto o PWT (page-level write through) acionam os pinos de saída de mesmo nome do processador para o controle de caches externas.

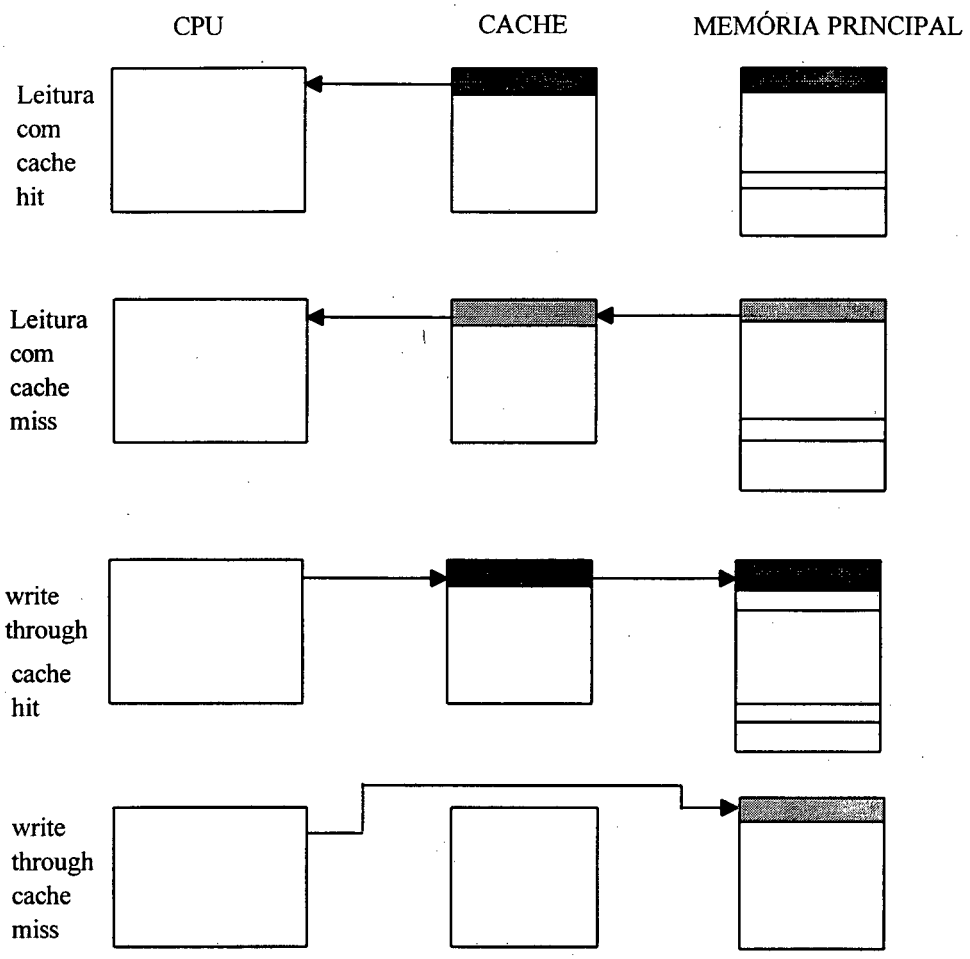


Figura B.18 - Comportamemto da cache write-through [INT87].

B.3.3.1. O Bit PCD

O bit PCD controla a "cacheabilidade" à nível de páginas. Quando uma entrada na tabela de páginas possui PCD=1 a cache está desabilitada para aquela página, mesmo que o hardware solicite a cache pela ativação da entrada KEN#. Quando este bit está zerado, a cache está habilitada para aquela página e pode ser solicitada via hardware ciclo a ciclo.

A desabilitação da cache é necessária para páginas que contêm ports de I/O mapeados em memória. Também é útil para páginas que não fornecem benefícios quando armazenadas em cache, como por exemplo, softwares de inicialização.

O bit PCD é mascarado pelo bit CD (cache disable) no registro de controle 0 (CR0). Se CD=1 o processador forçará o pino PCD sempre para alto. Se CD=0 o pino PCD terá o seu valor igual ao bit PCD da entrada das tabelas de páginas/diretórios.

B.3.3.2. O Bit PWT

Quando o bit PWT numa entrada da tabela de páginas, é igual a 1 , a política de write-through é aplicada para os dados da página correspondente. Se PWT=0, a política de write-back é aplicada. O estado de PWT é ignorado internamente pelo 80486 já que a cache interna é do tipo *write-through*. Caches externas, entretanto, podem usar o mecanismo *write-back* ou *write-through*, e portanto, podem usar o sinal de saída acionado pelo bit PWT para controlar a política da cache página a página (figura B.19).

Há 3 fontes de onde os bits PCD e PWT podem ser usados para um ciclo de barramento: do registrador CR3, do diretório de páginas e das tabelas de página. Estes bits são assumidos estarem em nível 0 durante o modo real, quando a paginação é desabilitada ou em ciclos onde a paginação não é utilizada (referências a I/O, reconhecimento de interrupção e ciclos de halt).

Quando a paginação é habilitada, os bits da entrada da tabela de páginas estão guardados no TLB (Translation Lookaside Buffer), e são usados quando a página mapeada pela entrada é referenciada. Nos ciclos de memória normais, PWT e PCD são lidos da entrada da tabela de páginas. Durante os ciclos de *refresh* do TLB, onde é realizada a leitura de uma nova entrada da tabela de páginas ou da tabela de diretório de páginas e sua colocação no TLB, os bits PWT e PCD são obtidos da tabela de diretório de páginas para ciclos de atualização da tabela de páginas, ou obtidos do registrador CR3 para ciclos de atualização da tabela de diretórios de páginas.

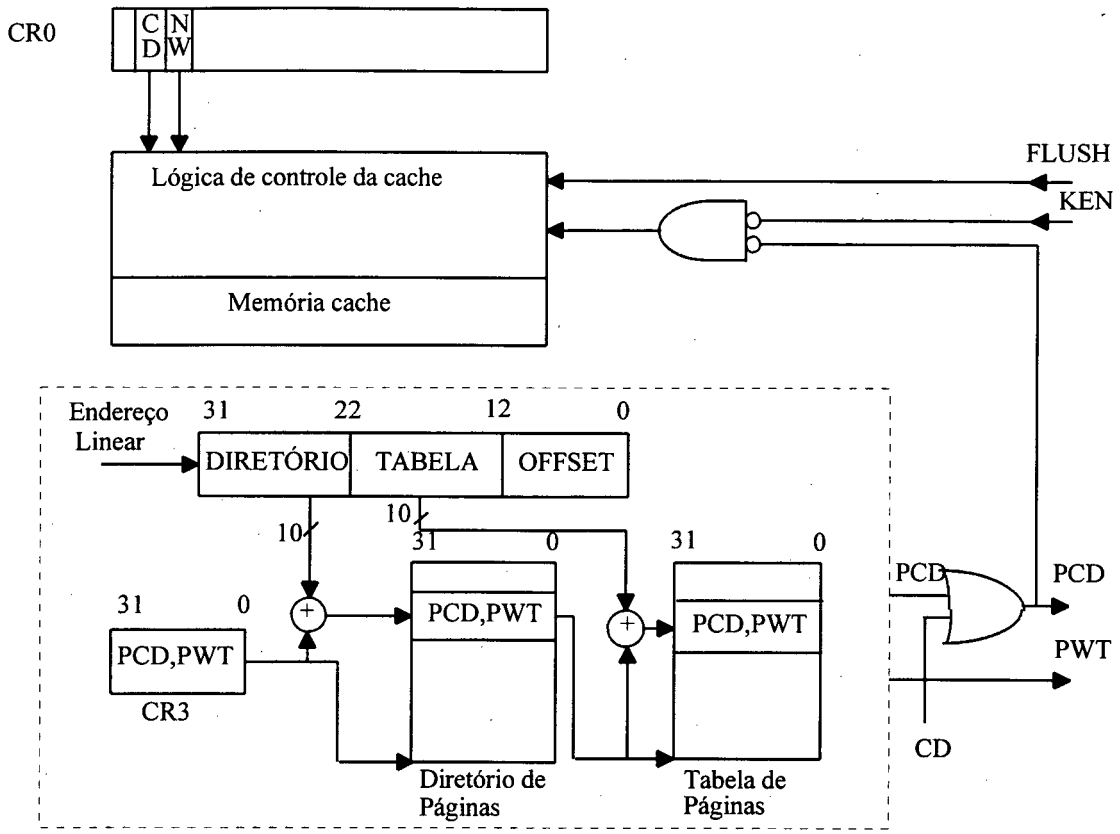


Figura B.19 - Esquema de controle da cache página a página [INT87].

O mecanismo de "cacheabilidade" por páginas permite estabelecer quais páginas (2 de 4Kbytes no máximo) podem ser armazenadas na cache para acesso rápido pelo processador. Colocando o bit PCD igual a 0 nas entradas da tabela de páginas para as duas páginas pré-definidas e setando-o nas demais é possível garantir que um trecho específico de um programa, ou um programa inteiro de no máximo 8Kbytes, seja sempre executado a partir da cache , ou seja, apresentando um alto grau de desempenho. A figura B.20 mostra um esquema de acesso à páginas "cacheáveis" e "não cacheáveis".

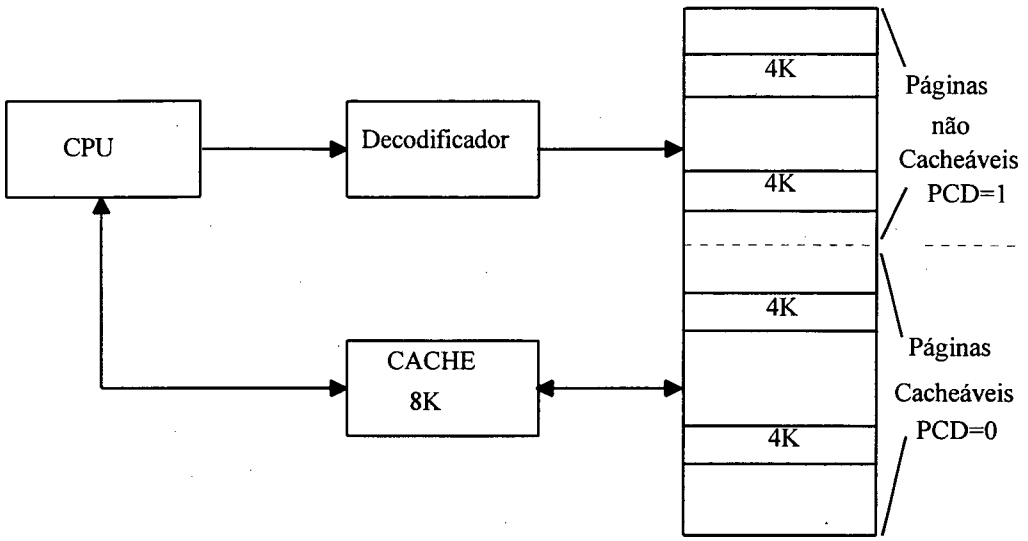


Figura B.20 - Esquema de acesso a páginas "cacheáveis" e "não cacheáveis".

B.3.4. Cache *Flushing*

Quando estas páginas não necessitam mais estar presentes na cache, pode-se usar uma instrução de software ou um sinal de hardware externo para resetar a cache (flushing). Este processo reseta todos os bits "validado" de todas as linhas da cache. As instruções INVD e WBINVD provocam o reset da cache interna. A cache externa, caso esteja presente, também é sinalizada para resetar seu conteúdo quando estas instruções são executadas. A diferença entre as instruções é que WBINVD sinaliza a cache externa do tipo *write-back* a escrever as linhas "sujas" na memória antes de resetar o seu conteúdo. Não há diferenças entre as duas instruções para a cache interna uma vez que a mesma é do tipo *write-through.*

B.3.5. Controle da Cache

Os bits CD e NW do registrador de controle 0 (CR0) controlam o modo de operação da cache. O bit CD habilita e desabilita a cache enquanto o bit NW controla a

operação de write-through e as invalidações dos dados da cache. A tabela B.2 descreve os 4 modos de operação possíveis usando os bits CD e NW.

CD	NW	Descrição
1	1	O preenchimento das linhas da cache e as invalidações são desabilitadas mas as linhas válidas presentes continuam respondendo a ciclos de memória. Esse modo permite a utilização da cache como uma memória RAM estática de alta velocidade já que pode-se carregar na cache as áreas de memória desejáveis no modo normal de operação e depois setar CD=1 e NW=1. Ao desligar-se a cache os seus dados contidos são "congelados". Para desabilitá-la completamente deve-se entrar nesse modo e executar um "flushing".
1	0	O preenchimento de linhas na cache é desabilitado mas a operação de write-through e invalidações são habilitadas mantendo a cache válida.
0	1	Modo inválido. Se CR0 é carregado com esses valores uma falha de proteção geral ocorrerá.
0	0	Operação normal da cache.

Tabela B.2 - Modos de operação da memória cache.

Através do entendimento dos modos de operação descritos na tabela acima pode-se executar o carregamento na cache de uma área de memória desejada através de sua leitura ou usando funções de testabilidade disponíveis na arquitetura do 80486. No instante em que os bits NW e CD forem colocados em nível alto, o conteúdo da cache é congelado possibilitando a leitura da área pré-estabelecida sempre a partir da cache, o que garante sua execução sempre com um alto desempenho.

B.4. Multitarefa

Os processadores 80386/80486 fornecem suporte de hardware para a implementação de multitarefa. Este recurso da arquitetura suporta diretamente a implementação dos sistemas operacionais multitarefa. Uma tarefa é um programa que está rodando ou esperando ser rodado enquanto outro é executado. Assim podemos ter operações tais como: edição de um arquivo, a compilação de outro, a transmissão de outro e a impressão de outro podendo ser executados como diferentes tarefas ao mesmo tempo em um sistema operacional ou em aplicações que utilizem or recursos de multitarefa [INT87]. Uma tarefa, teoricamente executa em paralelo com outras e na prática nos pequenos intervalos de tempo de execução (conhecidos como "time-slice", ou fatia de tempo) por parte de um processador.

Uma tarefa pode ser invocada por interrupção, exceção, desvio (jump) ou chamada (call). A transferência de execução feita por uma dessas formas permite que o estado da tarefa corrente seja armazenado antes do início da próxima tarefa. Isto é possível a partir de determinados tipos de segmentos e descritores especialmente relacionados com tarefas: segmentos de estado de tarefa (TSS), descritores de segmento de estado de tarefa e *gates* de tarefa.

B.4.1. O Segmento de Estado da Tarefa - TSS -

O segmento de estado da tarefa (TSS) é a estrutura de dados que contém todo o estado de execução de uma tarefa. A figura B.21 mostra o formato de um TSS para uma tarefa no 80386/80486. Neste segmento os campos são divididos em 2 categorias:

- Campos dinâmicos - aqueles que o processador atualiza a cada chaveamento de tarefas:
  - Registradores de segmento (ES, CS, SS, DS, FS e GS);
  - Registradores de uso geral (EAX, EBX, ECX, EDX, ESP, EBP, ESI e EDI);
  - Registrador de flags (EFLAGS);

- Ponteiro de instruções (EIP);

- Back link (seletor do TSS da tarefa anterior quando um retorno é esperado).

➢ Campos estáticos - aqueles que o processador lê mas não modifica:
- Seletor da LDT da tarefa;
- O endereço lógico dos stacks dos níveis de privilégio 0, 1 e 2 ;
- O bit T (debug trap bit) que, quando setado, ocasiona uma exceção de debug ao ocorrer um chaveamento;
- O endereço base do mapa de bits de permissão de I/O que inicia nos endereços mais altos do TSS.

Com esta estrutura o processador pode chavear entre tarefas salvando o contexto da tarefa atual no seu TSS e lendo o contexto da próxima tarefa no TSS correspondente. Assim, o estado da tarefa interrompida é guardado para quando ela for retomada, a execução recomeçe na instrução seguinte à última executada assim como, os registradores sejam carregados com os mesmos valores que tinham no momento da interrupção.

O TSS corrente é identificado por um registrador especial chamado registrador de TSS (TR). Esse registrador contém um seletor que referencia um descritor de TSS na GDT que por sua vez define o TSS corrente. O endereço base e o limite do TSS associado ao TR são informações escondidas e são automaticamente atualizados quando um novo seletor for carregado em TR.

O TSS pode ser acessado basicamente através de dois tipos de descritores: descritores de TSS e *gates* de tarefa.

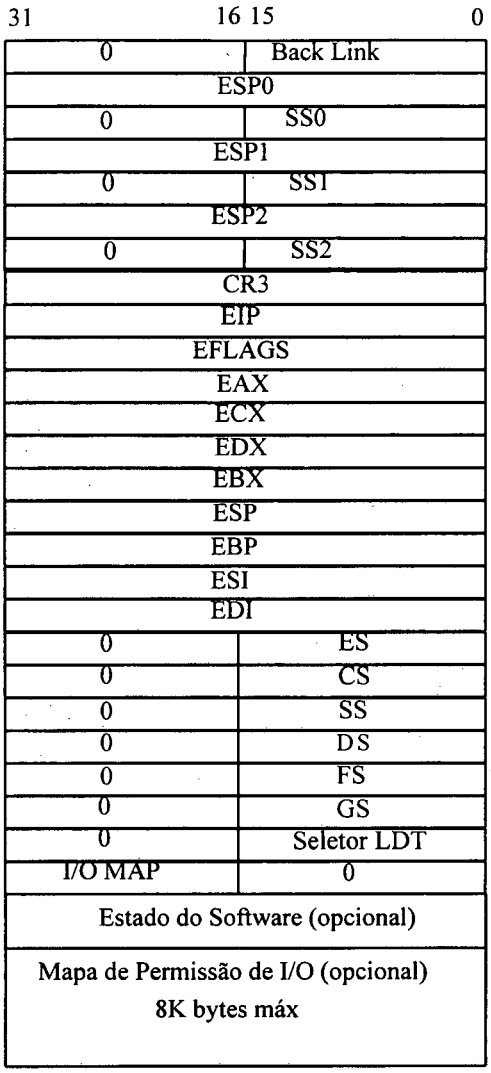
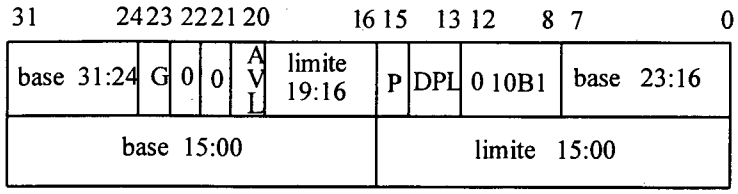


Figura B.21 - Segmento de Estado da Tarefa.

B.4.2. Descritor de TSS

O segmento de estado da tarefa (TSS), bem como todos os outros segmentos na arquitetura do 80386/80486, é definido por um descritor de segmento. A figura B.22 mostra o formato de um descritor de TSS.



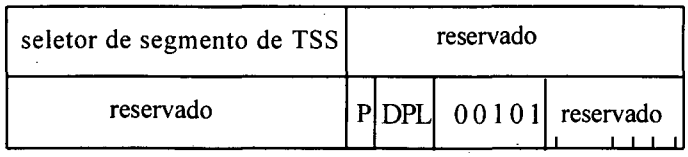
BASE Endereço base do segmento
DPL Nível de privilégio do descritor
LIMITE Limite do segmento
G Granularidade
P Segmento presente
AVL Disponível para uso pelo software do sistema
B Bit de ocupado

Figura B.22 - Descritor de TSS.

Os campos *DPL, base, limite, o bit de granularidade* e o *bit presente* têm funções similares aos usados nos descritores de segmento de dados. O campo *limite* deve ter valor maior ou igual a 67H, que é um byte a menos que o tamanho do TSS, porém pode ser maior se o mapa de permissão de I/O for usado. O *bit de ocupado* indica se uma tarefa está ocupada ou não. Uma tarefa é considerada ocupada se está executando ou esperando execução. Como as tarefa não são re-entrantes, a tentativa de chamar uma que esteja com o *bit de ocupado* setado causa exceção de proteção geral (número 13).

B.4.3. Descritor de Gate de Tarefa

O descritor de *gate* de tarefa é usado para permitir que tarefas menos privilegiadas possam chavear para tarefas mais privilegiadas. O campo seletor do *gate* indexa um descritor de TSS na GDT e o campo DPL controla o acesso a esse descritor. Uma tarefa só pode invocar um *gate* de tarefa se o seu CPL for numericamente menor ou igual ao DPL do descritor. Isto previne que tarefas menos privilegiadas causem um chaveamento indesejado. Quando *gates* de tarefa são usados para chaveamento o DPL do descritor de TSS destino é desconsiderado. A figura B.23 mostra um *gate* de tarefa.



DPL - menor nível de privilégio que uma tarefa deve ter para acessar o gate

SELETOR (16 bits) - seletor do segmento de estado da tarefa

Figura B.23 - Descritor de *gate* de tarefa.

B.4.4. Chaveamento de Tarefas

O 80386/80486 transfere a execução para outra tarefa em um dos seguintes casos:

- A tarefa atual executa uma instrução de JMP ou CALL para um descritor de TSS

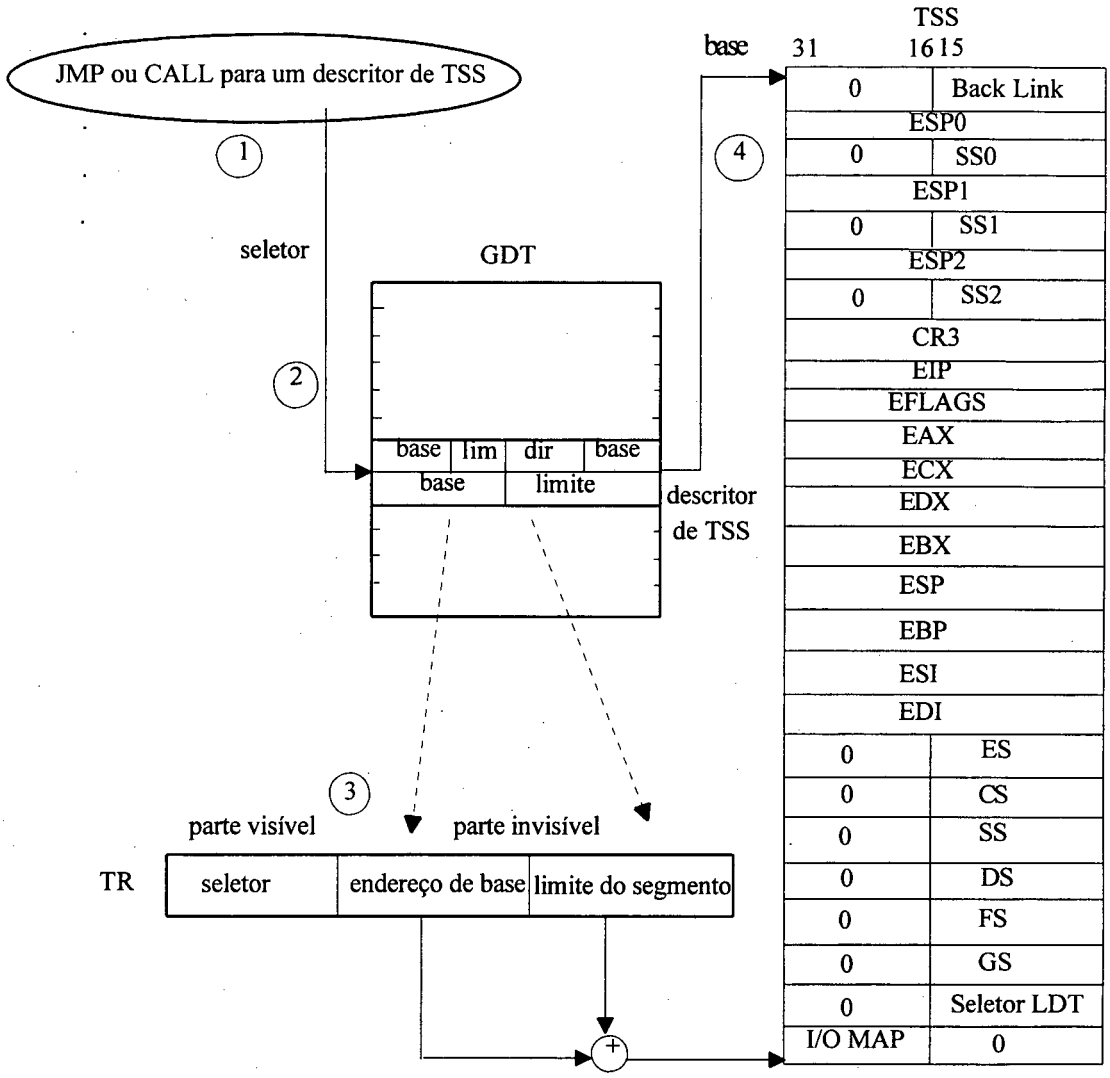


Figura B.24 - Chaveamento de tarefas usando JMP ou CALL para descritor de TSS.

- A tarefa corrente executa uma instrução de JMP ou CALL para um *gate* de tarefa

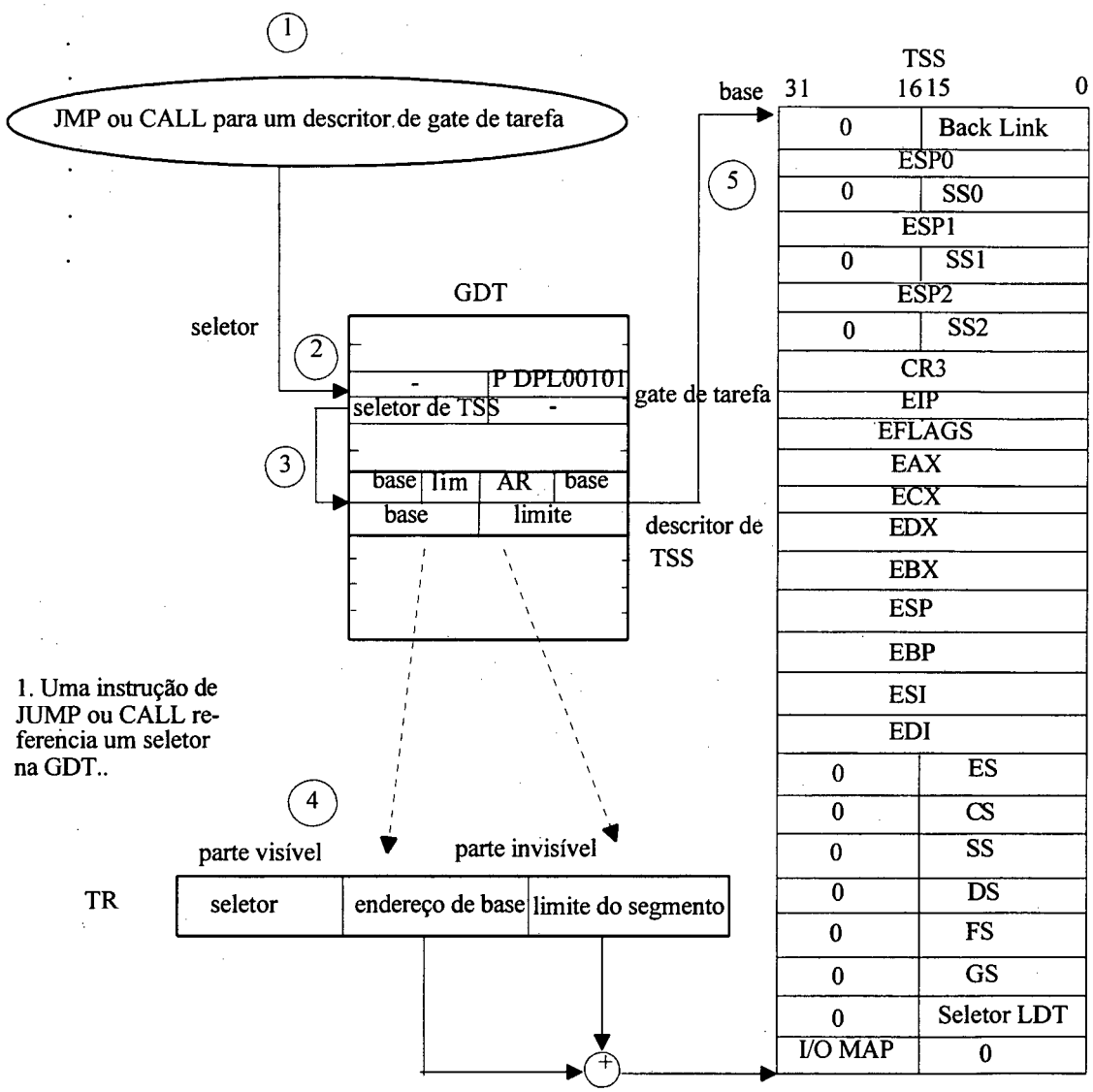
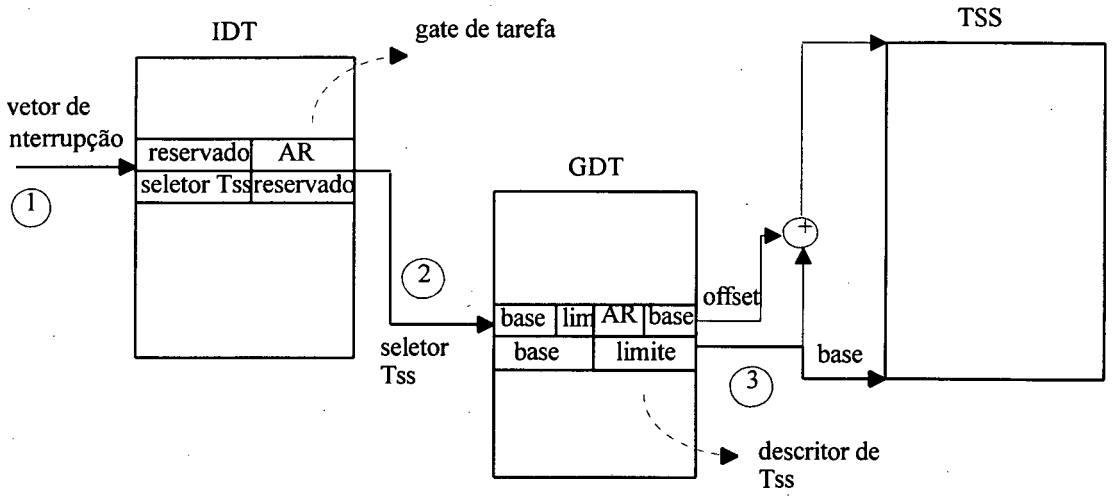


Figura B.25 - Chaveamento de tarefas usando JMP ou CALL para *gate* de tarefa.

- Uma interrupção ou exceção indexa um *gate* de tarefa na IDT



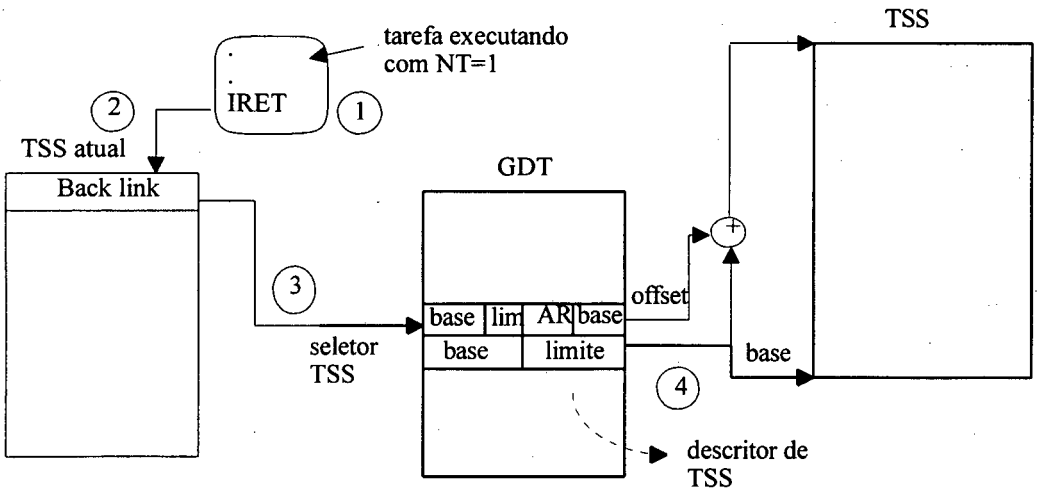
1. Na ocorrência de uma interrupção ou exceção um gate de tarefa é indexado na IDT.

2. O gate de tarefa indexa um descritor de TSS na GDT.

3. O Tss é acessado onde os valores dos registradores para a rotina de serviço (nova tarefa) de interrupção ou exceção são lidos.

Figura B.26 - Chaveamento de tarefas por interrupção ou exceção.

- A tarefa corrente executa um IRET com o flag NT setado



1. A tarefa corrente executa uma instrução IRET com NT=1.

2. O seletor de TSS no campo back link do TSS da tarefa atual é lido.

3. O seletor lido indexa um descritor de TSS na GDT.

4. O TSS é acessado onde os valores dos registradores para a rotina de serviço (nova tarefa) de interrupção ou exceção são lidos .

Figura B.27 - Chaveamento de tarefas por IRET.

Vários testes são feitos durante o chaveamento de tarefas, como por exemplo, se o descritor de TSS não está ocupado, se o segmento do TSS não é maior que o limite máximo de 103 bytes, assim como todos os seletores de segmento da nova tarefa são checados quanto à validade, presença, tamanho e regras de privilégio. Se alguma condição não for satisfeita uma exceção ocorrerá durante o chaveamento.

B.4.5. Aninhamento de Tarefas

O campo back link do TSS e o flag NT são usados para retornar a execução para a tarefa anterior. O flag NT indica se a tarefa atual está aninhada e, caso positivo (NT=1), o seletor do último TSS aninhado está armazenado no back link da tarefa atual.

Quando uma instrução CALL, uma interrupção ou exceção ocorre, o seletor do TSS da tarefa corrente é copiado no back link do TSS da nova tarefa e o flag NT é setado. Assim, quando a nova tarefa executar uma instrução IRET, o flag NT é testado e se estiver setado, o processador faz o chaveamento para a tarefa definida pelo seletor guardado no back link de seu TSS.

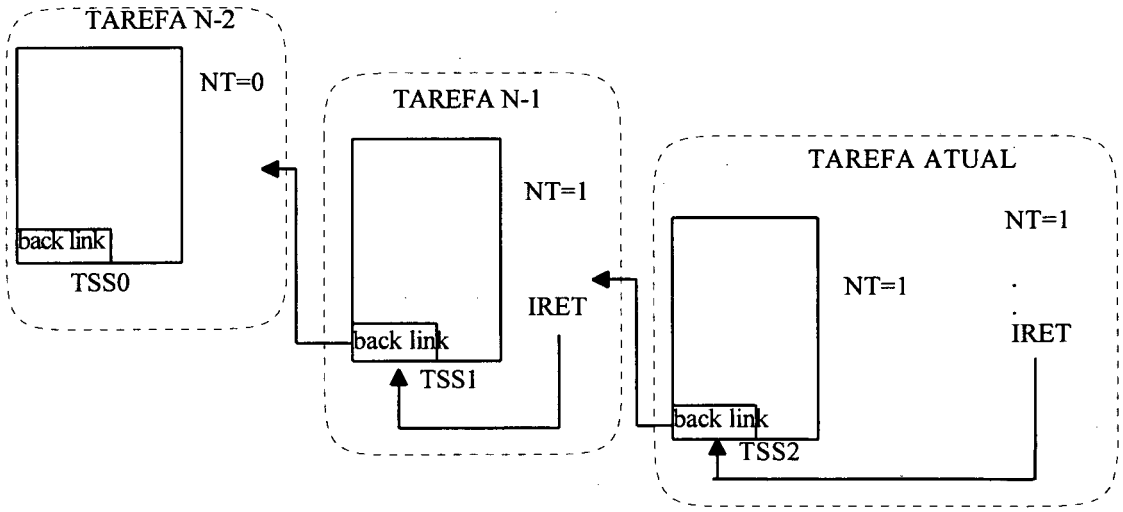


Figura B.28 - Aninhamento de tarefas.

Pode-se entretanto definir a tarefa para a qual se pretende chavear dentro do esquema de aninhamento mostrado na figura B.28. Se mudarmos o back link do TSS atual com o seletor de TSS da tarefa que desejarmos que seja executada em seguida e setarmos o flag NT, quando o IRET for executado ocorrerá o chaveamento para a tarefa desejada. Porém, é preciso notar que esta é única maneira de chavear para uma tarefa que tenha seu bit B (bit de ocupado) setado, qualquer modo diferente que através da instrução IRET causaria uma exceção durante o chaveamento. Para alterarmos o back link no TSS da tarefa corrente deve-se proceder da seguinte forma: carregar o registrador DS com um segmento de dados definido exatamente na mesma posição e de mesmo limite do TSS corrente - um alias do TSS. Feito isso, basta usar o DS para escrever o seletor de TSS desejado usando um offset igual a 0 - o campo back link ocupa a primeira palavra no TSS.

Exemplo:

```
MOV     AX, seletor _alias_TSS    ; move para AX o seletor do alias do TSS
MOV     DS,AX                     ; e passa para DS
MOV     AX, seletor_novo_TSS      ; move o seletor do TSS desejado para AX
MOV     WORD PTR DS:[0],AX        ; e escreve no back link do TSS atual
IRET                              ; chaveia para a tarefa estabelecida
```